uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería de Sistemas Audiovisuales
2017/2018

Trabajo de Fin de Grado

"Evaluación de protocolos limitados de nivel de aplicación para Internet de las Cosas"

Oscar Kowalewski Stempel

Tutora

María Celeste Campo Vázquez

Leganés, 18 de octubre de 2018

RESUMEN

En la actualidad es muy frecuente encontrar cada vez más dispositivos conectados a internet. La presente memoria trata de realizar un estudio sobre la situación del Internet de las Cosas y, más concretamente, de los protocolos de nivel de aplicación para este sector. La investigación se centra en los protocolos que se utilizan sobre redes y nodos con recursos limitados, haciendo especial hincapié en el protocolo CoAP.

En el estudio se expone de forma teórica el protocolo CoAP incluyendo la arquitectura, la implementación y los parámetros de este. Además del planteamiento teórico, se realiza una evaluación práctica del protocolo por medio de un sistema de peticiones y respuestas de cliente-servidor, desarrollado en una aplicación sobre la plataforma Arduino, que utiliza los datos de las condiciones ambientales obtenidas por un sensor. Para ello, se ejecuta una serie de cuatro experimentos, de los cuales el primero realiza una comparación de las prestaciones que ofrecen los protocolos HTTP y CoAP sobre el entorno descrito, y los tres restantes evalúan distintas funcionalidades que ofrece el protocolo de estudio. Las funcionalidades estudiadas son la diferencia entre mensajes enviados de forma fiable y no fiable, el descubrimiento de recursos gestionados por un servidor y la observación de un recurso. En todos los casos se ha basado en las características de tamaño del mensaje, tiempo de respuesta del servidor y consumo de energía para realizar la evaluación de las prestaciones.

Los resultados obtenidos demuestran que es necesario adaptar la arquitectura de internet a las nuevas formas de comunicación existentes, utilizando los protocolos adecuados para cada entorno. También destacan las distintas funcionalidades que ofrece el protocolo CoAP, el cual ha ido evolucionando a lo largo de los últimos años y posee una gran proyección de futuro en el sector del Internet de las Cosas.

Palabras clave: Internet de las Cosas; CoAP; protocolo de nivel de aplicación; Arduino; redes de sensores

ABSTRACT

Nowadays it is very common to find more and more devices connected to the internet. The current report tries to accomplish a study about the situation of the Internet of Things and, more specifically, of the application layer protocols for this sector. The research is focused on protocols that are used over constrained nodes and networks, and especially on the CoAP protocol.

In the investigation, a theoretical explanation of the CoAP protocol including the architecture, the implementation, and the parameters is done. In addition to the theoretical approach, a practical evaluation of the protocol is carried out by means of a client-server request-response system, developed in an application over Arduino platform, which uses the data of the weather conditions obtained by a sensor. Four experiments have been executed in order to reach the goal. The first one makes a comparison of the protocols CoAP and HTTP while the remaining three check different functionalities that the CoAP protocol offers. The functionalities studied in that case were the difference between the reliable and not reliable transmission, the resource discovery and the observation of a resource. In all the cases, the characteristics analyzed were the message length, the server response time and the power consumption.

The results of the experiments show that it is necessary to adapt the internet architecture to the new ways of communication, using the appropriate protocols for each environment. Also, it is important to highlight the different features that offer the CoAP protocol, which has been evolving throughout the last years and has a great future projection in the Internet of Things sector.

Key words: Internet of Things; CoAP; application-layer protocol; WSN; Arduino

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

Vivimos en un mundo en el que se tiende a que los objetos sean transformados para trabajar de forma autónoma, sin intervención humana, convirtiéndose en dispositivos que se pueden comunicar entre sí por medio de internet. Internet está formado por un conjunto de redes que permite la interconexión descentralizada de dispositivos a través de un grupo de protocolos. Existen diferentes modelos en internet que siguen una arquitectura basada en capas, y cada uno de ellos describe cómo es el proceso de envío de datos a través de una red.

El Internet de las Cosas es el presente y futuro de los próximos años en el sector de las TIC y surge la necesidad de estudiar una forma más adecuada para la comunicación entre estos aparatos. Aunque es cierto que para adaptar la comunicación al entorno de IoT sería necesaria una optimización en todos los niveles, no es tan sencillo realizar un cambio de la arquitectura de internet al completo. Es por ello que el enfoque principal se toma en torno a los protocolos de nivel de aplicación, ya que a este nivel se manifiestan las principales diferencias a la hora de procesar los datos y generar los paquetes, teniendo en cuenta que en IoT los dispositivos tienen recursos limitados.

El presente Trabajo de Fin de Grado se centra en la evaluación de las prestaciones del protocolo de nivel de aplicación CoAP. CoAP es un protocolo destinado para ser usado en dispositivos con recursos limitados como los presentes en las redes de sensores inalámbricos. Se trata de un protocolo que pretende replicar el modo de funcionamiento de HTTP, pero diseñado para poder trabajar en aplicaciones M2M e IoT. Para realizar la evaluación, se trata de comparar en un mismo escenario de Internet de las Cosas si el protocolo CoAP ofrece un mejor rendimiento que el protocolo HTTP.

Además, al tratarse de un sector en constante desarrollo, es habitual que aparezcan nuevos protocolos que pretenden mejorar las prestaciones durante el intercambio de mensajes. Por este motivo, se realiza asimismo una recopilación sobre la situación actual del Internet de las Cosas, teniendo en cuenta los distintos protocolos que han repuntado más en las últimas décadas, como MQTT o XMPP, y que, por tanto, ofrecen mejores servicios para el sector de interés.

1.1. Motivación

Internet ha experimentado una gran propagación durante las últimas décadas. Tal es la magnitud de esta expansión, que el sector de las telecomunicaciones busca facilitar la vida de las personas por medio de la interconexión y automatización de los dispositivos de uso cotidiano.

Este fenómeno, denominado Internet de las Cosas, tiene grandes ventajas para la sociedad, pero también algunas taras. Una de estas taras es la baja eficiencia de la

arquitectura existente actualmente para el Internet de las Cosas a nivel de aplicación, ya que los protocolos empleados en internet no ofrecen las mismas prestaciones en IoT.

El protocolo utilizado por excelencia para la navegación web, HTTP, no es el ideal para dispositivos con recursos limitados, por ello surge la necesidad de recurrir a otros protocolos que se adapten a los requisitos de estos dispositivos. El protocolo escogido para este propósito es CoAP, ya que tiene la ventaja de que está diseñado con la posibilidad de traducirse a HTTP, permitiendo la integración de los dispositivos con la web. De la misma manera que HTTP realiza peticiones a una web API, CoAP obtiene los datos de un sensor y, como ambos protocolos se basan en el modelo REST, de ahí que se haya escogido el protocolo HTTP como referencia en este proyecto.

1.2. Objetivos

El objetivo principal de este Trabajo de Fin de Grado es analizar el protocolo de nivel de aplicación de internet CoAP y evaluar su uso en un escenario sencillo dentro del entorno del Internet de las Cosas.

A raíz de este objetivo global, se definen los siguientes objetivos específicos:

- Estudiar el estado del arte de las tecnologías pertenecientes al sector del Internet de las Cosas.
- Analizar teóricamente el protocolo CoAP.
- Diseñar de un escenario único para la realización de las pruebas.
- Ejecutar las pruebas sobre los protocolos CoAP y HTTP.
- Evaluar los resultados obtenidos.
- Redactar la memoria.

1.3. Estructura de la memoria

- En el primer capítulo de la memoria, además de realizar una breve introducción para contextualizar el Trabajo de Fin de Grado, se indica la motivación, los objetivos y la presente estructura de la memoria.
- En segundo lugar, se encuentra el estudio del estado del arte, donde se describe el sector del Internet de las Cosas, la necesidad de un cambio en el protocolo IP y el nacimiento de distintos protocolos de nivel de aplicación que han destacado en el ámbito del IoT.

- En el tercer capítulo se define en detalle el protocolo CoAP, el cual pretende mejorar la comunicación de dispositivos de recursos limitados. Para ello se describe la arquitectura del protocolo, su implementación y los parámetros principales.

- En cuarto lugar, se encuentra la parte práctica del proyecto. En este apartado se define el escenario sobre el que se han realizado las pruebas y los resultados obtenidos de las mismas.

- En el quinto capítulo se hace referencia al marco regulador sobre el que se encuentra el IoT y los estándares técnicos que engloban al protocolo CoAP.

- En el sexto capítulo se describe el presupuesto del TFG y el impacto socio-económico de la aplicación del mismo.

- En el séptimo capítulo se exponen los objetivos cumplidos, las líneas de trabajo futuras y las conclusiones del proyecto.

- Por último, se encuentran los anexos que contienen las imágenes respectivas a los componentes utilizados, el glosario y el Extended Abstract de la presente memoria.

2. ESTADO DEL ARTE

2.1. Introducción al Internet de las Cosas

El Internet de las Cosas (IoT, por sus siglas en inglés) [1][2] es un concepto que hace referencia a la interconexión de objetos cotidianos con internet. Si bien es verdad que la existencia de dispositivos autónomos interconectados entre sí es bastante antigua (por ejemplo, los cajeros automáticos aparecieron en 1967), el término de *Internet of Things* no aparece hasta el año 1999 de la mano de Kevin Ashton. Tuvieron que pasar 6 años cuando, en el 2005, la ITU mostró interés en este concepto realizando un reporte exhaustivo.

Se trata de un sector en auge, que pretende automatizar las labores más cotidianas del día a día para así facilitar la vida a las personas. Este fenómeno ha tenido gran éxito en el mundo de la domótica, consiguiendo casas conectadas casi en su totalidad a internet, que pueden: abrir y cerrar puertas, subir y bajar persianas, regular la calefacción o el aire acondicionado, encender y apagar las luces, y un largo etcétera. Todos estos eventos controlados, por ejemplo, desde el móvil, sin necesidad de estar en la propia casa.

Más allá del ámbito doméstico se pueden encontrar en multitud de sectores, como en la hostelería, con dispositivos en cadenas de comida rápida que avisan cuando la comida está lista; en sanidad, con herramientas que permiten un control de los pacientes; en logística, donde se pueden realizar seguimientos del lugar en el que se encuentra un pedido en cada momento; o en agricultura y ganadería, con sensores de temperatura, humedad y demás factores climáticos. Este último servicio es el que se pretende reproducir en los casos prácticos de este proyecto.

Además, es importante mencionar que el Internet de las Cosas está relacionado con otros servicios, cuyo crecimiento impacta directamente en este sector. Se trata de las siguientes áreas:

- Big Data: se trata de la gestión y análisis de los datos que no son posibles de capturar y procesar por medio de las herramientas convencionales ya que superan los límites de las mismas. Hay tres características fundamentales que definen el Big Data, llamadas las tres V's: velocidad, volumen y variedad.

  La importancia del Big Data reside en que la cantidad de datos que se generan a diario es imposible de gestionar por los métodos convencionales y surge la necesidad de utilizar nuevas herramientas para analizarlos. La recopilación y el estudio de los datos permite responder a las respuestas de las empresas e incluso a trazar tendencias.

- Machine Learning: es una rama de la inteligencia artificial cuyo objetivo reside en el aprendizaje automático a partir de ejemplos, es decir, en generar algoritmos a partir de unos datos de entrada para predecir comportamientos

futuros. Además, estos sistemas se van actualizando con el tiempo en función de los datos suministrados sin necesidad de la intervención humana.

- Ciberseguridad: consiste en el conjunto de medidas que se toman para asegurar la protección de cualquier elemento conectado a internet, ya sean datos o dispositivos, y la recuperación de los mismos en caso de producirse un ataque.

Los cambios acontecidos en internet durante las últimas décadas conllevan al mismo tiempo un cambio en la estructura de este. Teniendo en cuenta que los dispositivos que trabajan sobre el Internet de las Cosas poseen unas características limitadas en cuanto a la potencia o el ancho de banda, la arquitectura debe implementar protocolos que se adapten a estos escenarios restringidos. En especial, se han visto afectados dos niveles: el nivel de red y el nivel de aplicación.

2.2 IPv6

Debido a la expansión de internet, tanto geográfica como en cuanto a la variedad de dispositivos que pueden utilizarlo, el protocolo de red IPv4 se vio limitado cuando sus más de 4 mil millones de posibles direcciones se iban agotando a ritmos acelerados. A raíz de este despliegue, el organismo encargado de la estandarización de internet, el IETF, vio la necesidad de reemplazar este protocolo por una versión más moderna que satisficiese el incremento de usuarios en internet.

El protocolo IPv6 [3] es la versión más reciente de IP que pretende reemplazar a su predecesor IPv4 y tiene la capacidad para identificar hasta 2^{128} direcciones, es decir, más de 340 sextillones. Esto se consigue por medio de direcciones que pasan de estar formadas por 32 bits a 128 bits, usando un formato de ocho grupos de cuatro dígitos hexadecimales:

1234:5a8e:23b2:0dd6:876b:ae42:55d7:cba0

La importancia de IPv6 para el Internet de las Cosas reside en 2 características esenciales. En primer lugar, se encuentra el aumento del soporte de usuarios previamente comentado, ya que es necesaria una gran cantidad de direcciones para los dispositivos autónomos y es imposible visualizar un futuro para IoT en un escenario limitado como el de IPv4. En segundo lugar, IPv6 cuenta con una mayor seguridad, incorporando cifrado y verificación de autenticidad del origen de los mensajes, muy importante cuando hablamos de conectar un coche autónomo o una *smart city* a internet y abrir así la puerta a la posibilidad de un hackeo.

2.2.1. 6LoWPAN

Aplicado al Internet de las Cosas, surge el estándar 6LoWPAN (*IPv6 over Low-Power Wireless Personal Area Network*) [4], una capa de adaptación que permite a las redes

basadas en el estándar de la capa de nivel físico IEEE 802.15.4 utilizar el protocolo IPv6. El objetivo de este estándar es el de facilitar la interconexión de los dispositivos más pequeños, de escasa potencia y con recursos limitados. Para lograr la integración de estos dispositivos, se realiza una compresión de la cabecera de IPv6, ya que los paquetes enviados sobre el protocolo IEEE 802.15.4 pueden tener un tamaño máximo de 81 bytes y sólo la cabecera de IPv6 tiene un tamaño fijo de 40 bytes.

El escenario para el que está diseñado 6LoWPAN es el de las aplicaciones que requieren una conexión inalámbrica utilizando dispositivos de baja potencia. Tiene especial utilidad en redes de sensores inalámbricos o WSN en las que el objetivo es monitorizar y recolectar la información respectiva a las condiciones ambientales de un entorno mediante lecturas de un conjunto de sensores.

2.3. Protocolos destacados de IoT

En este punto se tiene en cuenta los protocolos de nivel de aplicación que han supuesto un despunte dentro del sector del Internet de las Cosas. El principal interés de estos protocolos es que son el lugar donde se produce la encapsulación de los datos, y como su nombre indica, se generan de distintas formas en función del uso que se le vaya a dar a esos datos. Se han escogido los protocolos MQTT, XMPP y CoAP debido a su impacto y se realiza un análisis de su arquitectura y la manera en que trabajan.

2.3.1. MQTT

MQTT (*Message Queuing Telemetry Transport*) [5] es un protocolo de internet de nivel de aplicación orientado a comunicaciones M2M en el Internet de las Cosas. Se trata de un protocolo que trabaja sobre TCP/IP y permite el cifrado de mensajes confidenciales usando una conexión con TLS. Está diseñado para minimizar el ancho de banda y el uso de recursos gracias a intercambios de mensajes extremadamente ligeros. El protocolo fue inventado por Andy Stanford-Clark de IBM y Arlen Nipper de Arcom en el año 1999 convirtiéndose en estándar de OASIS en el año 2014.

Para lograr una carga reducida en sus mensajes utiliza un sistema de publicación-suscripción. Este sistema trabaja de manera que los publicadores, es decir, los emisores de los mensajes no programan el mensaje para que sea enviado al receptor, sino que categorizan los mensajes en distintos temas sin conocer los suscriptores a los que se enviará. Del mismo modo, los suscriptores muestran interés en distintos temas desconociendo el origen de los mismos.

Este protocolo requiere de un “*broker*”, que es un nodo central que hace las labores de servidor para gestionar el intercambio de mensajes desde el emisor hasta el receptor o receptores que estén suscritos al tema de interés.

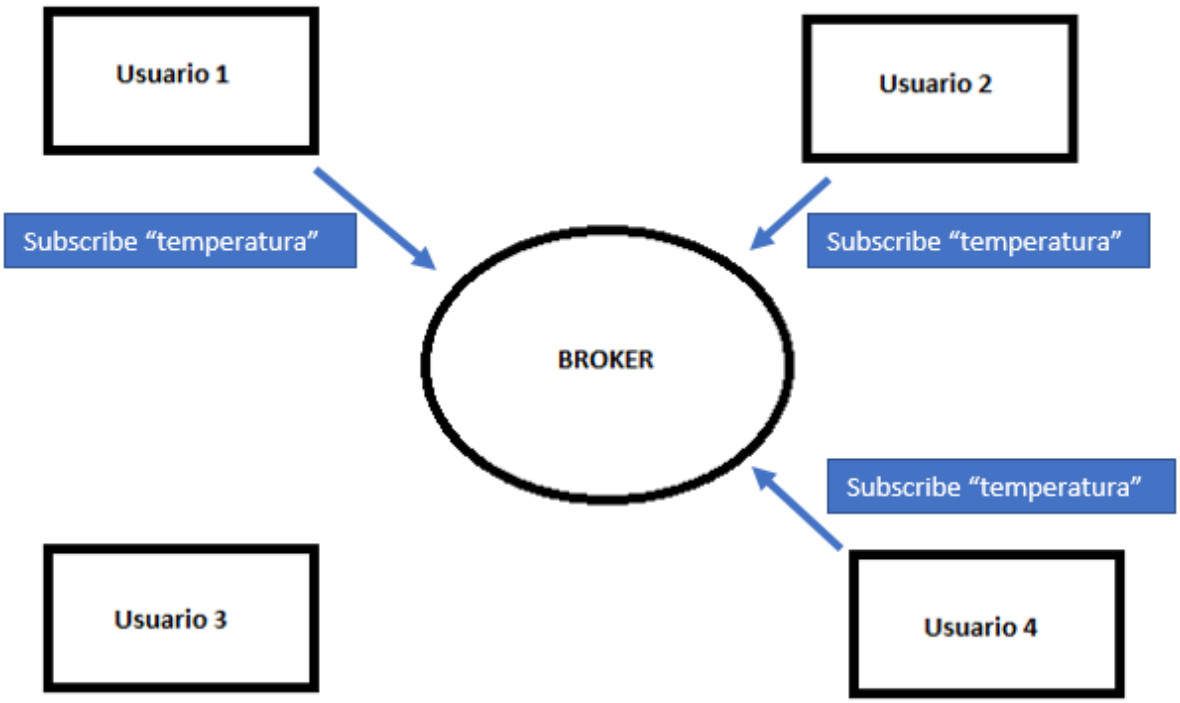

Figura 1: Suscripción en el protocolo MQTT

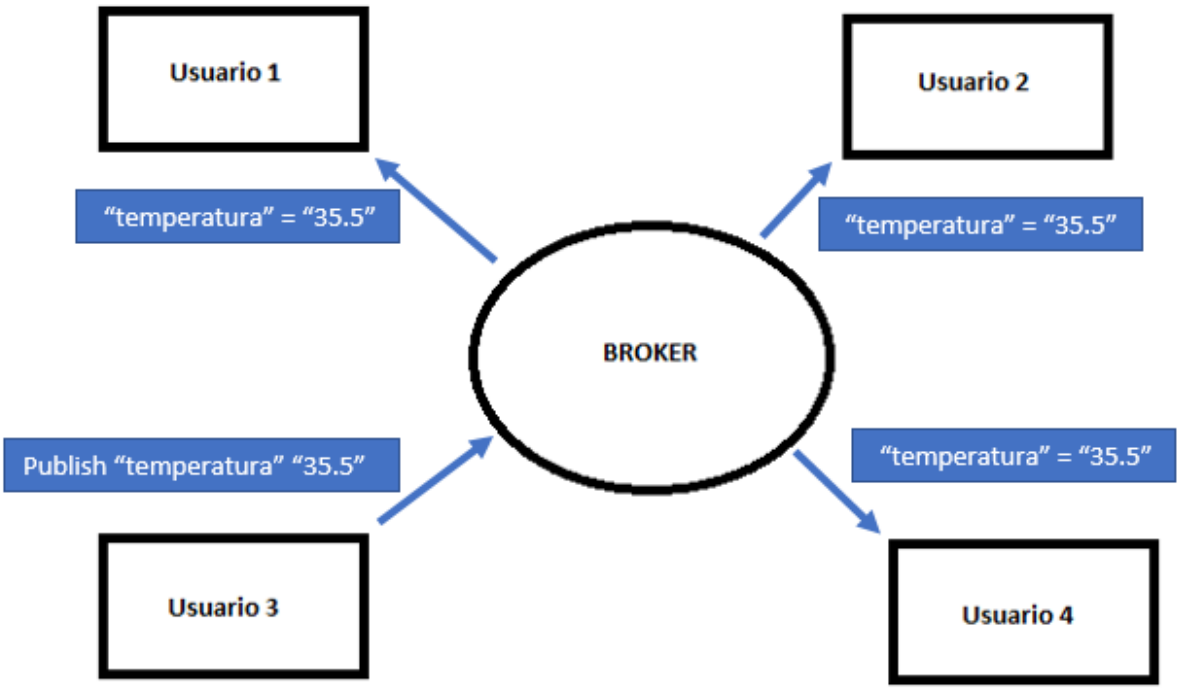

Figura 2: Publicación en el protocolo MQTT

Los temas se organizan en una jerarquía con forma piramidal. De esta forma, los suscriptores pueden escoger la información concreta que desean recibir.

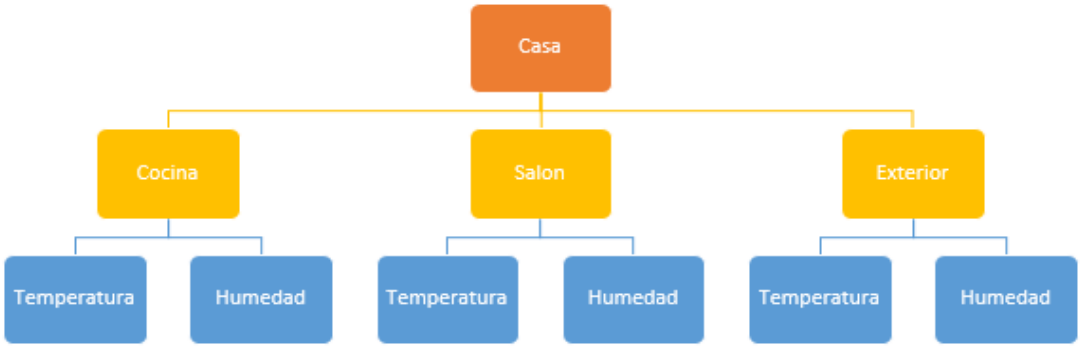

Figura 3: Estructura de los temas del protocolo MQTT

Así, si un suscriptor estuviese interesado en la temperatura que hay en el exterior de la vivienda, tendría que suscribirse a "Casa/Exterior/Temperatura" o, si estuviese interesado en todos los datos referentes al salón, lo haría por medio de "Casa/Salon/#".

2.3.2. XMPP

XMPP (*Extensible Messaging and Presence Protocol*) [6] es un protocolo de nivel de aplicación que surgió originalmente como medio para el intercambio de mensajes instantáneos, es decir, para servicios de chat. Este protocolo surgió de la mano de Jeremie Miller en el año 1999 con el nombre de "*Jabber*", aunque actualmente está en desuso. Se trata de un protocolo basado en XML y, al igual que HTTP y MQTT, trabaja sobre TCP/IP. XMPP ha obtenido un gran éxito en el sector de la mensajería instantánea, llegando a ser utilizado por aplicaciones como WhatsApp o Facebook.

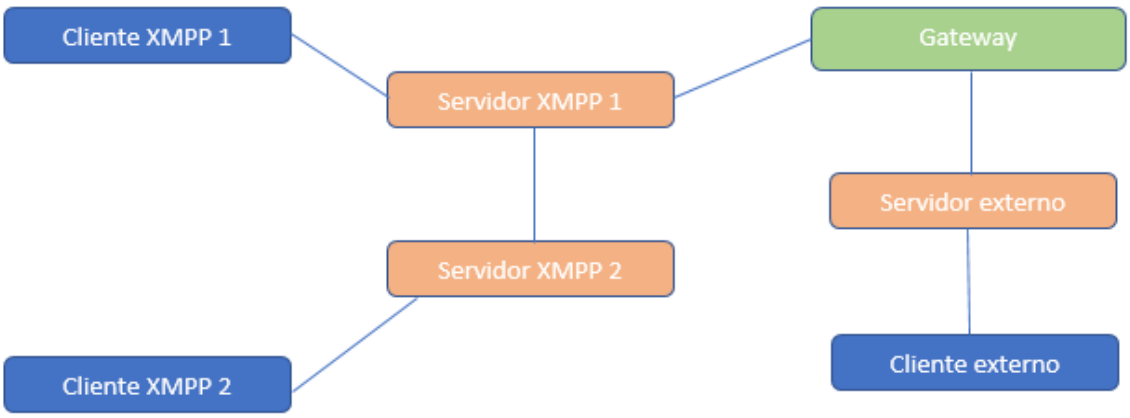

Figura 4: Arquitectura del protocolo XMPP

En cuanto a la arquitectura, el protocolo está compuesto por 3 elementos: los clientes de XMPP, los servidores de XMPP y los gateway. Para que los clientes de XMPP se pongan en contacto entre ellos, deben recurrir a los servidores de XMPP, los cuales se encargan de realizar el intercambio de los mensajes. Si un cliente de XMPP se quiere comunicar con un cliente que no utiliza el protocolo XMPP, los servidores XMPP

encaminan los mensajes hacia un gateway que traducirá el mensaje a otro protocolo distinto soportado por el cliente de destino.

El funcionamiento de XMPP se basa en dos conceptos fundamentales: *XML Streams* y *XML Stanzas*. En primer lugar, los *XML Streams* son los encargados de establecer la conexión entre dos entidades y, una vez establecida, se envían los *XML Stanzas*. Los *XML Stanzas* pueden ser de tres tipos

- *message*: se utiliza para enviar un mensaje instantáneo entre dos clientes XMPP.

- *presence*: este modo tiene como finalidad enviar mensajes de tipo broadcast a todos los clientes que estén suscritos

- *iq* (info/query): se trata de un intercambio de mensajes en el que un cliente le solicita información a otro.

El protocolo XMPP tiene tres configuraciones posibles o, dicho de otro modo, tres maneras de intercambiar mensajes y, dependiendo del modo en el que se configure la conexión, se enviará un tipo de *XML Stanza*, que son: petición/respuesta (*iq*), publicación/suscripción (*presence*) y mensaje asíncrono (*message*).

2.3.3. CoAP

CoAP (*Constrained Application Protocol*) [7] es un protocolo de internet diseñado especialmente para la comunicación M2M (máquina a máquina). Los dispositivos a los que está destinado este protocolo pertenecen a la rama del Internet de las Cosas y se caracterizan por un bajo consumo de recursos y una implementación sencilla, centrado sobre todo en los sectores de la energía inteligente y los elementos de automatización.

El uso de servicios web en internet ha llegado a ser omnipresente en la mayoría de las aplicaciones y depende de la estructura web fundamental REST originaria de HTTP. Es por ello por lo que CoAP trabaja sobre entornos limitados siguiendo el estándar CoRE (*Constrained RESTful Environments*) [8] de IETF, el cual proporciona un *framework* para aplicaciones orientadas a recursos y tiene como objetivo implementar la arquitectura REST sobre los nodos (por ejemplo, microcontroladores de 8 bits con RAM y ROM limitados) y redes (por ejemplo, 6LoWPAN) más restringidos. La arquitectura de CoRE está formada por nodos que son responsables de uno o varios recursos, como sensores o controladores de estados, y son los encargados del intercambio de mensajes.

Además, CoRE posee una serie de mecanismos que permiten descubrir los recursos que gestiona un dispositivo. Para ello, el cliente realiza una petición incluyendo el texto *"/.well-known/core*" al servidor destino, el cual responderá con un resumen de los recursos que gestiona, pero no de los valores de esos recursos. Gracias a este mensaje, el cliente es capaz de generar la URI para realizar la petición de un recurso en concreto de forma correcta al servidor.

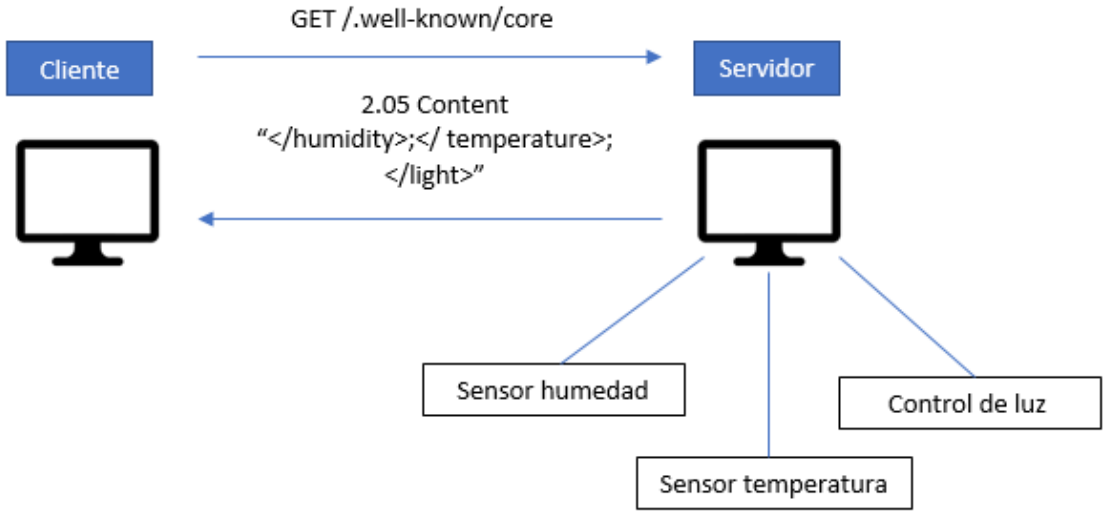

Figura 5: Funcionalidad de descubrimiento de CoRE

La finalidad del diseño de CoAP es mantener la carga de los mensajes pequeña, limitando así la necesidad de fragmentación. Esto no significa que se pretenda comprimir ciegamente HTTP sino más bien realizar un subconjunto de REST (POST, GET, PUT y DELETE) común a HTTP, pero optimizado para aplicaciones máquina-a-máquina. Por lo tanto, uno de los objetivos principales de CoAP es conseguir un protocolo web genérico que cumpla los requisitos de los entornos restringidos.

A diferencia de HTTP y los otros protocolos de IoT descritos previamente, CoAP maneja los intercambios de mensajes de forma asíncrona sobre un transporte orientado a datagramas, es decir, sobre el protocolo UDP. Este intercambio se produce de forma lógica usando una capa de mensajes que soporta una transmisión fiable. Otra diferencia respecto a MQTT y XMPP es que el intercambio de mensajes se realiza directamente entre un cliente y un servidor sin necesidad de elementos intermedios en la comunicación.

2.4. Conclusiones

Si bien es cierto que tanto MQTT con el modelo de *publish/suscribe*, como los distintos *XML Stanzas* que ofrece XMPP, son dos variantes interesantes, se ha escogido el protocolo CoAP como nodo central de este Trabajo de Fin de Grado. Esta elección se debe a la variedad de funcionalidades que ofrece el protocolo CoAP manteniendo un esquema conocido como es la arquitectura REST.

En los capítulos siguientes se realiza un análisis teórico más completo y una evaluación práctica del protocolo CoAP. Además, en la parte práctica se ha decidido comparar el protocolo CoAP con el protocolo HTTP, debido a que el primero está diseñado para mantener la arquitectura REST utilizada en HTTP, pero en dispositivos y redes más restringidos. Este diseño permite una integración en la web con el protocolo HTTP por lo que es muy interesante realizar una evaluación de ambos protocolos en un entorno del Internet de las Cosas.

3. EL PROTOCOLO COAP

3.1. Modelo de mensajería

CoAP define cuatro tipos de mensajes: Confirmable (CON), No Confirmable (NON), Asentimiento (ACK) y Reinicio (RST). Los Códigos de Método y los Códigos de Respuesta incluidos en algunos de estos mensajes son los que se encargan de hacerles llevar solicitudes o respuestas.

CoAP utiliza cabeceras cortas de longitud fija de 4 bytes que pueden ir seguidas de opciones y/o de carga útil. Este formato de mensaje lo comparten tanto las solicitudes como las respuestas. Cada mensaje contiene un identificador (*Message ID*) que es utilizado para detectar duplicados y para la opción de transmisión fiable.

La opción de transmisión fiable se consigue escogiendo el mensaje como Confirmable. Un mensaje Confirmable se transmite utilizando un tiempo de espera por defecto y un tiempo de reenvío exponencial entre retransmisiones sucesivas hasta que el receptor envía un mensaje de Asentimiento con el mismo identificador y este llega al emisor. Cuando el receptor no es capaz de procesar del todo el mensaje Confirmable, responde con un mensaje de Reinicio en lugar de uno de Asentimiento.

Cualquier mensaje que no requiera de una transmisión fiable puede ser enviado como un No Confirmable. Estos mensajes son respondidos con otro mensaje No Confirmable, y no con Asentimientos, pero si tienen identificador para detectar los mensajes duplicados, y también responden con un Reinicio si no son capaces de procesar el mensaje de origen.

Como CoAP funciona sobre UDP, también soporta el uso de direcciones de IP de múltiple destino, permitiendo solicitudes CoAP de tipo multicast.

3.1.1. Modelo de solicitud/respuesta

Las semánticas de solicitud y respuesta de CoAP son llevadas en mensajes que incluyen los Códigos de Método y los Códigos de Respuesta respectivamente. La información opcional, como la URI o la carga útil, se transporta como una opción de CoAP. Se utilizan Tokens para emparejar las respuestas con las solicitudes independientemente de los mensajes subyacentes.

Las solicitudes pueden ser transmitidas en mensajes Confirmables o No Confirmables y, si están disponibles inmediatamente, las respuestas a los mensajes Confirmables se envían en el ACK resultante junto con la respuesta, en lugar de enviar sólo el ACK. A esta técnica de respuesta se la denomina *piggybacking*.

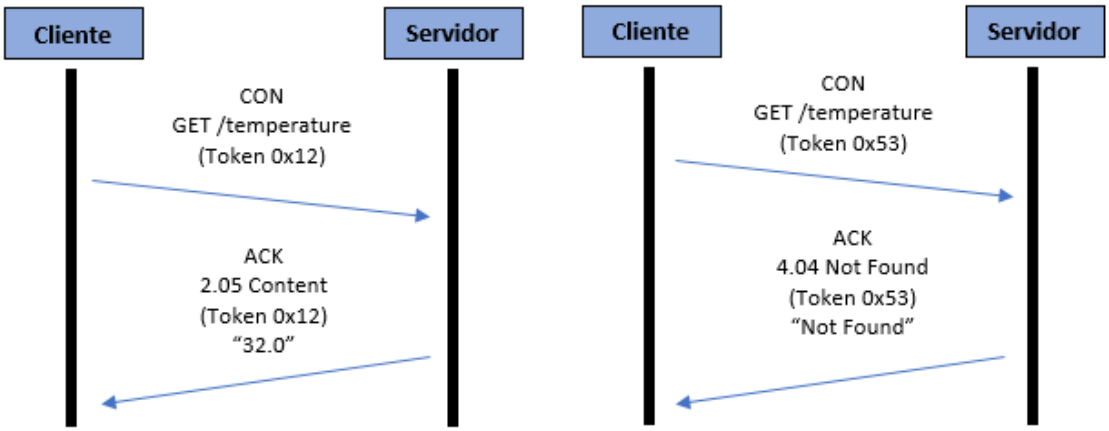

Figura 6: Solicitudes GET con respuestas "piggybacked" en CoAP

Si el servidor no es capaz de responder inmediatamente a una solicitud de tipo Confirmable, simplemente responderá con un mensaje ACK vacío para que el cliente pueda parar de retransmitir la solicitud. Una vez que la respuesta está lista, el servidor la envía en un nuevo mensaje Confirmable, el cual deberá ser contestado en este caso por el cliente con un Asentimiento.

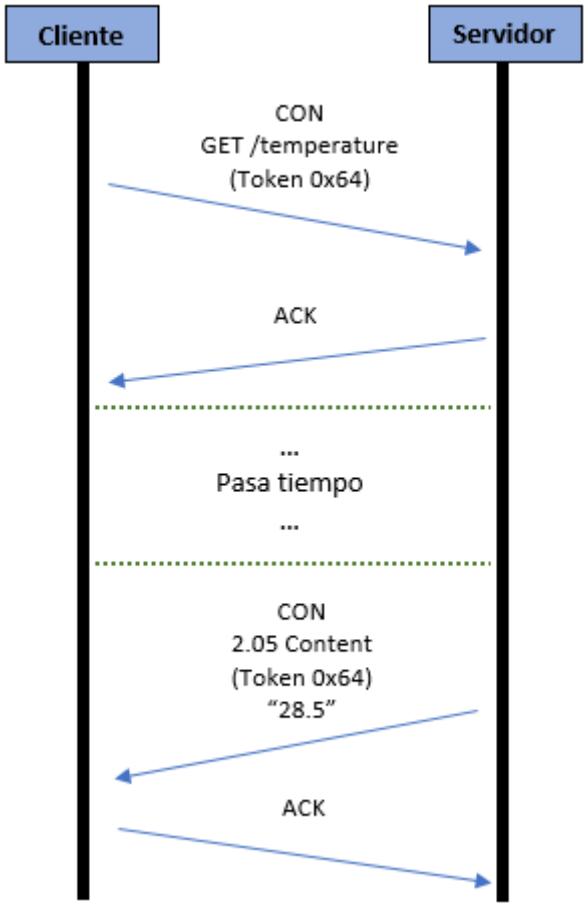

Figura 7: Solicitud GET que no puede ser respondida inmediatamente en CoAP

Por otro lado, si el cliente realiza una solicitud por medio de un mensaje No Confirmable, entonces la respuesta que se enviará será también de tipo No Confirmable.

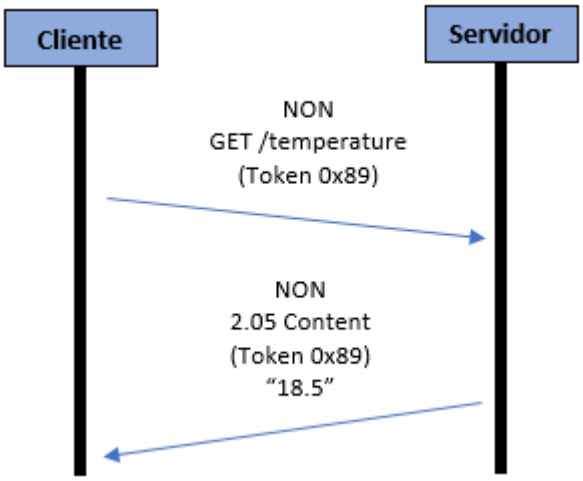

Figura 8: Solicitud GET por medio de un mensaje No Confirmable en CoAP

Además de las respuestas indicadas en los casos anteriores, existe la posibilidad de que el receptor no sea capaz de procesar la solicitud, en cuyo caso, enviará como respuesta un Reinicio. El mensaje RST indica que se ha recibido el mensaje, pero falta algún componente para poder interpretarlo. Un caso común es cuando el receptor se ha reiniciado y por lo tanto olvida alguna condición que pudiese requerirse para procesar el mensaje.

3.2. Formato del mensaje

Los mensajes de CoAP están codificados con un formato binario simple. La trama comienza con una cabecera de tamaño fijo de 4 bytes, seguida de un Token de tamaño variable entre 0 y 8 bytes, y por último las opciones y la carga útil, ambas opcionales.

Ver	T	TKL	Code	Message ID
Token				
Opciones				
11111111			Carga Útil	

Figura 9: Formato del mensaje de CoAP

Los campos en la cabecera corresponden a la primera fila del mensaje y se distribuyen de la siguiente manera:

- Versión (Ver - 2 bits): Indica la versión de CoAP, actualmente se debe poner a este campo el valor de 1 ya que sólo existe esta opción. Otros valores están reservados para versiones futuras y los mensajes que los contengan actualmente serán ignorados.

- Tipo (T - 2 bits): Indica si el mensaje es de tipo Confirmable (0), No Confirmable (1), Asentimiento (2) o Reinicio (3).

- Tamaño del Token (TKL - 4 bits): Indica el tamaño, entre 0 y 8 bytes, del Token. Los tamaños entre 9 y 15 bytes están reservados y no deben ser enviados puesto que darán lugar a un error de formato.

- Código (Code - 8 bits): Los 3 bits más significativos corresponden a la clase del mensaje y puede tomar valores entre 0 y 7, si bien actualmente sólo se utilizan los de solicitud (0), respuesta exitosa (2), respuesta de error del cliente (4) y respuesta de error del servidor (5). Los 5 bits restantes equivalen a los detalles del mensaje y comprenden los valores desde el 00 hasta el 31.

  El código al completo sigue el formato “c.dd” (c-clase, dd-detalle). El valor 0.00 indica que se trata de un mensaje vacío; entre el 0.01 y el 0.31, ambos incluidos, están las solicitudes asignadas por los Códigos de Método de CoAP; entre el 2.00 y el 5.31 se encontrarán las respuestas asignadas por los Códigos de Respuesta de CoAP. El resto de los valores (1.00-1.31 y 6.00-7.31) están reservados. Todos los posibles códigos de Método y Respuesta existentes actualmente en CoAP se pueden encontrar en la página de IANA dentro de los parámetros de CoRE [9].

- ID del mensaje (Message ID - 16 bits): Se trata del identificador del mensaje y se utiliza para detectar mensajes duplicados y para emparejar las solicitudes (CON/NON) con sus respectivas respuestas (ACK/RST).

La cabecera va seguida del valor del Token, el cual tendrá el tamaño especificado previamente en la cabecera y se utiliza para relacionar las solicitudes con las respuestas. Después del Token va una determinada cantidad de opciones que pueden ser incluidas en el mensaje. Cada instancia de opción contiene las características de número de la opción definida en CoAP a la que se refiere, el tamaño del valor de la opción y el valor en sí. En lugar de especificar el número de opción directamente, las instancias deben de aparecer ordenadas por el número al que se refieren y se usa una codificación Delta entre ellas. El número de cada instancia se obtiene sumando su Delta con el número de opción de la instancia anterior (siendo este valor 0 en la primera instancia). Si se utilizan varias instancias de la misma opción, se deben tomar deltas con valor 0.

Los campos de las opciones se distribuyen de la siguiente forma:

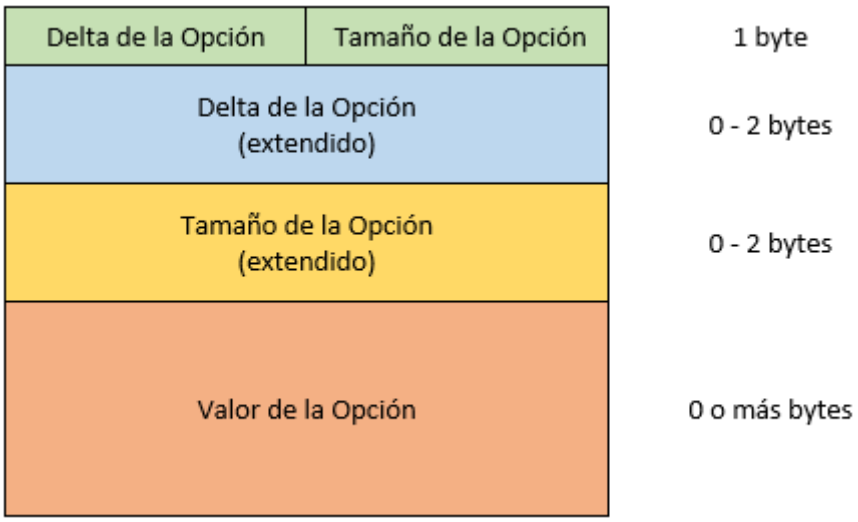

Figura 10: Formato de las opciones de CoAP

- Delta de la opción (4 bits): El valor de la Delta de la opción se usa como diferencia entre el número de opción de la instancia actual y la anterior o, lo que es lo mismo, el número de opción es igual a la suma de la Delta de la opción de la instancia actual más la de todas las anteriores.

- Tamaño de la opción (4 bits): Como el propio nombre indica, hace referencia al tamaño del valor de la opción en bytes.

- Tanto la Delta como el tamaño de la opción extendidos se utilizan únicamente en casos especiales para los valores reservados 13, 14 y 15.

- Valor: El campo del valor de la opción puede tener los siguientes formatos:

    1. Vacío: Una secuencia de bytes de tamaño 0.

    2. Opaco: Una secuencia de bytes opaca.

    3. Uint: Un número entero no negativo representado a partir del número de bytes especificados en el campo del tamaño. Se deben evitar los 0 a la izquierda que no aporten ningún valor, es decir, si el número a enviar es el 1 por ejemplo, se envía un solo byte con ese valor y no una secuencia de bytes 0000001.

    4. String: Una cadena de caracteres codificada en UTF-8.

Por último, después de las opciones se puede encontrar la carga útil. En caso de existir, para fijar un límite entre el final las opciones y el comienzo de la carga útil, se utiliza un prefijo de un byte denominado Marcador de Carga Útil con el valor 0xFF (en la figura 9 se representa en el campo previo a la carga útil - 11111111). La información de la carga útil ocupa el espacio comprendido entre el marcador y el final del datagrama de UDP.

3.3. Transmisión de mensajes

Los mensajes de CoAP se transmiten de forma asíncrona entre puntos finales. Como el protocolo está limitado a transportes poco fiables como UDP, los mensajes pueden llegar desordenados, aparecer duplicados o perderse sin previo aviso. Por este motivo, CoAP implementa un mecanismo de confianza ligero, sin intentar recrear todo el transporte como sucede en TCP. La transmisión tiene las siguientes características:

- Una retransmisión simple de parada y espera, cuyo tiempo de espera crece exponencialmente después de paradas sucesivas en mensajes de tipo Confirmable.

- Detección de duplicados tanto para mensajes Confirmables como No Confirmables.

3.3.1. Puntos finales

Un punto final corresponde a la fuente o el destino de un mensaje de CoAP. En función del modo de seguridad usado para mandar el mensaje, NoSec, PreSharedKey, RawPublicKey o Certificate, el punto final se identificará de una manera u otra. Así, si se escoge el modo no seguro (NoSec), el punto final estará identificado por una dirección IP y un número de puerto de UDP. Los modos seguros implementan DTLS (protocolo que proporciona seguridad en las comunicaciones con protocolos de datagramas) y cada uno ellos es el responsable de proporcionar el identificador al punto final.

3.3.2. Correlación de mensajes

Los mensajes de tipo ACK y RST están relacionados con los Confirmables y No Confirmables por medio de un identificador de mensaje (*Message ID*) junto con información adicional correspondiente a la dirección del punto final. El identificador del mensaje se genera en el remitente y el destinatario debe implementar ese mismo identificador para contestar a los mensajes recibidos. Este identificador estará reservado durante un periodo de tiempo para uso exclusivo de una única solicitud y respuesta (incluye a los mismos mensajes retransmitidos varias veces).

3.3.3. Mensajes duplicados

Un destinatario puede recibir el mismo mensaje Confirmable varias veces, por ejemplo, cuando se pierde el mensaje de ACK o no llega al remitente antes de que termine el temporizador. En estos casos, el destinatario debe responder a todos los

mensajes Confirmables duplicados utilizando el mismo ACK o RST pero procesando la solicitud o respuesta sólo una única vez.

Ahora, si el mensaje que se recibe duplicado es de tipo No Confirmable, sólo la primera solicitud o respuesta es procesada y respondida, mientras que todos los demás se ignoran.

3.3.4. Tamaño del mensaje

La especificación de CoAP proporciona sólo un límite superior para el tamaño de mensaje. Los mensajes mayores que un paquete IP dan como resultado una indeseable fragmentación de paquete. Un mensaje de CoAP, debidamente encapsulado, debería caber en un sólo paquete de IP y dentro de una carga útil de UDP.

Estas elecciones de tamaño para CoAP funcionan bien sobre IPv6 y la mayoría de IPv4 actuales. No engloba a todas las IPv4 debido a que no es seguro al 100% de que, en este caso, no haya fragmentación de paquetes.

3.3.5. Control de congestión

El control de congestión es proporcionado por el mecanismo de aumento de tiempo exponencial de retransmisión entre envíos sucesivos.

Para evitar la congestión, los clientes (incluyendo a los proxies) deben limitar el número de interacciones simultáneas que mantienen con un servidor a un valor que por defecto es uno, pudiendo cambiar según el entorno. Se incluyen en estas interacciones los mensajes CON a los que no ha llegado todavía un ACK pero se espera que llegue o una solicitud para la que ni una respuesta ni un ACK han sido recibidos, pero se espera que lleguen.

3.4. Semánticas de solicitud y respuesta

CoAP opera sobre un modelo de solicitud/respuesta similar al que utiliza HTTP: el punto final en la función de “cliente” envía una o más solicitudes de CoAP a un punto final “servidor”, el cual procesa las solicitudes y envía respuestas. A diferencia de HTTP, estos mensajes no se mandan sobre una conexión previamente establecida, sino que son intercambiados de forma asíncrona sobre mensajes de CoAP.

3.4.1. Solicitudes

CoAP soporta las solicitudes por medio de los métodos GET, PUT, POST y DELETE. Una solicitud se inicia ajustando el campo del Código como Código de Método en la

cabecera de los mensajes Confirmables y No Confirmables e incluyendo la información de la solicitud. Estos métodos, al igual que en HTTP, pueden tener las propiedades de "*safe*" o "*idempotent*". Un método "*safe*", sólo atribuible a GET, no puede ejecutar otra acción sobre el recurso que no sea la de recuperación. Por el contrario, los métodos que utilizan la opción "*idempotent*" son GET (puede utilizar las dos opciones), PUT y DELETE. El método POST es un caso especial puesto que su efecto lo determina el servidor de origen y depende del recurso al que va destinado. Normalmente resulta en la creación de un nuevo recurso o la actualización del recurso de destino.

Descripción de los métodos:

- GET: recupera una representación para la información que corresponde actualmente al recurso identificado por la URI de la solicitud.

- POST: solicita que se procese la representación incluida en la solicitud.

- PUT: solicita que el recurso identificado por la URI de la solicitud sea actualizado o creado (si no existe) con la representación incluida.

- DELETE: solicita que se elimine el recurso identificado con la URI solicitada.

3.4.2. Respuestas

Después de recibir e interpretar una solicitud, un servidor contesta con una respuesta de CoAP que se enlaza por medio de un Token generado por el cliente. Hay que tener en cuenta que este enlace es distinto al que se produce con los identificadores de mensaje entre los mensajes Confirmables y su ACK.

Al igual que en el caso de las solicitudes, un mensaje de respuesta se identifica ajustando el campo de Código de la cabecera como Código de Respuesta. Similar a como procede HTTP, las respuestas de CoAP indican el resultado de la solicitud.

Como ya se comentó previamente, las respuestas tienen códigos definidos para las diferentes situaciones. Así, los tipos de respuesta pueden ser:

- 2.XX - Satisfactorio: La solicitud ha sido recibida correctamente, entendida y aceptada.

- 4.XX - Error de cliente: La solicitud contiene una sintaxis incorrecta o no puede cumplirse.

- 5.XX - Error de servidor: El servidor falló al realizar una solicitud aparentemente válida.

Las respuestas del servidor pueden enviarse de tres formas distintas:

- *Piggybacked*: esta es la contestación más frecuente y es transportada en un mensaje de tipo ACK que responde a un mensaje Confirmable.

- *Separate*: en los casos en los que no es posible responder con un *piggybacked*, como en los que el servidor necesita más tiempo para obtener la representación del recurso solicitado. Una manera de implementar esto, es iniciando el proceso para obtener la representación del recurso y poner un temporizador para mandar el ACK. El servidor puede también enviar inmediatamente un ACK si sabe de antemano que no podrá responder con un *piggybacked*. En ambos casos el ACK es una promesa de que la solicitud será respondida más tarde.

  Cuando el servidor finalmente obtiene la representación del recurso, envía la respuesta. Si se requiere que este mensaje no se pierda, entonces es enviado en un mensaje Confirmable desde el servidor al cliente, escogiendo el primero un nuevo identificador de mensaje.

- *Non-confirmable*: en caso de que la solicitud fuese encapsulada en un mensaje de tipo No Confirmable, la respuesta debería ser enviada del mismo modo.

3.4.3. Token

Los Tokens son usados para emparejar respuestas con sus respectivas solicitudes. Su valor es una secuencia de 0 a 8 bytes y todos los mensajes deben llevar uno. Cada solicitud encapsula un Token generado por el cliente que el servidor debe repetir en cualquiera que sea su respuesta. Se forman de tal manera que en una determinada pareja de remitente/destinatario el valor debe de ser único.

3.4.4. Opciones

Tanto las solicitudes como las respuestas pueden incluir una lista de una o más opciones. Estas opciones pueden ser clasificadas en dos clases: críticas u optativas. La diferencia entre ambas reside en cómo se maneja una opción que no es reconocida por el punto final. Así, las opciones no reconocidas de la clase optativa simplemente son ignoradas, mientras que los de la clase crítica tienen distintos comportamientos según el tipo de mensaje:

- Si se trata de una solicitud Confirmable provocará la devolución de una respuesta de 4.02 (opción mala).

- Si se trata de una respuesta Confirmable, un *piggybacked* en un ACK o un mensaje No Confirmable, el mensaje será rechazado.

Además de marcar las opciones como críticas u optativas, se pueden clasificar también en función de cómo un proxy hace frente a la opción si no la reconoce. Así, se puede distinguir entre las opciones "*Unsafe-to-forward*" (no seguro para avanzar) y

"*Safe-to-forward*" (seguro para avanzar) dependiendo si *UnSafe* está activo o no respectivamente.

Las distintas opciones de CoAP existentes pueden observarse en la página de IANA en la sección correspondiente a los parámetros de CoRE.

3.4.5. Almacenamiento en caché

Los puntos finales de CoAP pueden almacenar en caché las respuestas para reducir el tiempo de respuesta y el consumo de ancho de banda. El objetivo de este almacenamiento es reutilizar un mensaje de respuesta anterior para satisfacer una solicitud actual. Existen dos mecanismos mediante los cuales se pueden reutilizar componentes del caché:

- Modelo de frescura: Cuando se desea reutilizar respuestas almacenadas sin necesidad de una solicitud de red, reduciendo así la latencia y los viajes de ida y vuelta por la red. Para ello se utiliza un temporizador con la opción de "Max-Age" por la cual la respuesta se almacena en caché durante un tiempo máximo. Tras este tiempo se considera la respuesta como no "*fresca*".

- Modelo de validación: Cuando se desea reutilizar la carga útil de un mensaje anterior para satisfacer una nueva solicitud, reduciendo el consumo de ancho de banda de la red. En este caso, se tienen respuestas almacenadas, pero de carácter no fresco. El cliente puede utilizar la opción "ETag" en las solicitudes de tipo GET para permitir al servidor tanto seleccionar respuestas almacenadas para utilizarlas, como actualizar su frescura.

3.4.6. Proxying

Un proxy es un punto final de CoAP que puede ser asignado por los clientes de CoAP para realizar solicitudes en su nombre. Hay dos tipos de proxies, los "forward-proxy", que son explícitamente seleccionados por el cliente y los "reverse-proxy", que pueden ser insertados en los servidores de origen. Generalmente, un proxy necesita una forma de determinar los posibles parámetros que requiere una solicitud para un determinado destino en base a la solicitud que recibe del cliente. Esta forma está completamente definida para los forward-proxy, pero en los reverse-proxy depende de la configuración específica que tenga. En particular, el cliente de un reverse-proxy no indica el localizador para el destino, necesitando alguna manera de traducir el campo del nombre en el proxy.

Si el proxy no utiliza caché, simplemente envía la solicitud traducida al destino en cuestión. Por otro lado, si emplea caché pero no tiene una respuesta almacenada que enlace la solicitud traducida, entonces necesita actualizar su caché.

En el caso de los mensajes enviados por medio de forward-proxies, las solicitudes son enviadas mediante mensajes Confirmables o No Confirmables al proxy, pero difieren

frente a un sistema normal en que estos mensajes deben especificar la URI de destino final por medio de un string en la opción de Proxy-Uri o Proxy-Scheme.

Los reverse-proxies no utilizan las opciones anteriores, pero necesitan determinar el destino de una solicitud desde la información en la solicitud y la información en su configuración.

3.5. Las URIs de CoAP

CoAP utiliza los esquemas de URI “coap” y “coaps” para identificar los recursos y proporcionar medios para localizarlos. Los recursos están organizados de forma jerárquica y gobernadas por un servidor de origen que escucha solicitudes de CoAP (“coap”) o de CoAP protegido por DTLS (“coaps”) en un puerto de UDP determinado. La URI completa queda de la siguiente manera:

- Para “coap” ⇒ `"coap://" host [":" port] path-abempty ["?" query]`

- Para “coaps” ⇒ `"coaps://" host [":" port] path-abempty ["?" query]`

Si en el campo de host se proporciona una IP literal o una dirección IPv4, entonces el servidor de CoAP puede ser alcanzado con esa dirección IP. Si el campo del host es un nombre registrado, entonces ese nombre es considerado un identificador indirecto y el punto final puede utilizar un servicio de resolución de nombre, como DNS, para encontrar la dirección de ese host.

En ningún caso el campo del host puede ir vacío, y si se recibe, será considerado inválido. El subcomponente *port* indica el puerto UDP en el que está localizado el servidor CoAP. Si no se envía el puerto o está vacío, se asume el puerto por defecto 5683 para “coap” y el 5684 para “coaps”.

El “*path*” indica el recurso dentro del alcance del host y del puerto. Consiste en una secuencia de segmentos de ruta separados por barras oblicuas (“/”).

La query sirve para parametrizar aún más el recurso. Consiste en una secuencia de argumentos separados por un ampersand (“&”). Los argumentos habitualmente siguen el formato de “clave = valor”.

Además de las especificaciones previas, hay varias reglas que se siguen para normalizar la URI:

- Si el subcampo del puerto es igual al de por defecto, lo normal es omitirlo.

- Una componente de *path* vacía equivale a una ruta absoluta de “/”.

- Tanto el esquema como el host no distinguen entre mayúsculas y minúsculas.

- Los caracteres distintos a los reservados son equivalentes a sus bytes codificados de porcentaje.

Así, los siguientes ejemplos de URIs darían lugar al mismo mensaje de CoAP:

```
coap://example.com:5683/~sensors/temp.xml
coap://EXAMPLE.com/%7Esensors/temp.xml
coap://EXAMPLE.com:/%7esensors/temp.xml
```

3.6. Multicast de CoAP

CoAP permite hacer solicitudes a grupos de IP sobre multicast. Las solicitudes, que son de tipo No Confirmable, están caracterizadas por ser transportadas en un mensaje de CoAP que tiene como destino una IP multicast en lugar de un punto final. Para evitar una implosión de respuestas de error, cuando un servidor es consciente de que una solicitud de tipo multicast ha llegado, este no debe devolver un RST en respuesta al mensaje. Los mensajes de multicast sólo pueden ser transportados sobre UDP sin hacer uso del protocolo de datagramas DTLS. Esto significa que los modos de seguridad definidos para CoAP no son aplicables en multicast.

3.7. Seguridad de CoAP

Como ya se adelantó previamente, se pueden diferenciar entre distintos modos de seguridad. El modo NoSec es el único que no comprende ningún tipo de seguridad mientras que los demás utilizan DTLS. Se puede distinguir entre los siguientes modos seguros:

- PreSharedKey: hay una lista de claves compartidas previamente, y cada una de las claves indica los nodos que se pueden utilizar para transmitir el mensaje. En el extremo habrá una clave por cada nodo sobre el que CoAP necesite pasar. Por el contrario, si dos entidades comparten la misma clave, esta clave permitirá identificarlas como un mismo grupo.

- RawPublicKey: el dispositivo posee un par de claves asimétricas sin certificado que son validadas utilizando un mecanismo fuera de banda. Además, tiene un identificador calculado por medio de la clave y una lista de los identificadores de los nodos con los que se puede comunicar.

- Certificate: el dispositivo posee un par de claves asimétricas con certificado X.509 que lo enlaza a su sujeto y es firmado por alguna raíz de confianza común. Además, el dispositivo tiene una lista de pilares de confianza que puede usar para validar un certificado.

Durante la fase de aprovisionamiento, el dispositivo de CoAP es provisto de la información de seguridad que necesita, incluyendo los materiales para las claves y las listas de control de acceso. Al final de esta fase, el dispositivo estará en uno de los cuatro modos de seguridad.

Al igual que HTTP está asegurado mediante TLS sobre TCP, CoAP está asegurado usando datagramas de TLS (DTLS) sobre UDP. La diferencia entre los dos sistemas de seguridad es que DTLS tiene añadidas características para hacer frente a la naturaleza poco fiable del transporte sobre UDP.

El punto final que actúa como cliente de CoAP debe actuar también como cliente de DTLS. Para establecer una conexión, el cliente inicia la sesión con el servidor en el puerto apropiado y, cuando se hayan terminado de establecer los parámetros para la comunicación, el cliente puede comenzar con las solicitudes de CoAP.

3.8. Protocolo cruzado entre CoAP y HTTP

En ciertas ocasiones, se necesitará que CoAP se comunique con HTTP, y como CoAP soporta una cantidad limitada de funcionalidades de HTTP, el protocolo entre ambos debe ser sencillo. El protocolo en cuestión es llevado a cabo por un proxy intermedio que se encarga de traducir los mensajes recibidos y enviarlos al destinatario. Hay dos posibles direcciones para acceder al recurso y se realiza por medio de forward-proxies:

- Proxy CoAP-HTTP: Permite a los clientes de CoAP acceder a los recursos en servidores HTTP a través de un intermediario. Este proceso es iniciado incluyendo la opción de Proxy-Uri o Proxy-Scheme con una URI “http” o “https” en una solicitud de CoAP a un proxy CoAP-HTTP.

- Proxy HTTP-CoAP: Permite a los clientes de HTTP acceder a los recursos en servidores CoAP a través de un intermediario. Este proceso es iniciado especificando la URI como “coap” o “coaps” en la línea de petición de una solicitud de HTTP a un proxy HTTP-CoAP.

4. EVALUACIÓN PRÁCTICA

4.1 Diseño de pruebas

Para la parte práctica del Trabajo de Fin de Grado se ha optado por realizar una comparación entre los protocolos CoAP y HTTP. El punto de partida ha sido el artículo "*Evaluation of Constrained Application Protocol for Wireless Sensor Networks*" [10], en el cual se elabora una comparación entre estos dos protocolos. El escenario consiste en un servidor formado por un sensor de temperatura y humedad usado desde un dispositivo con el sistema operativo Contiki OS y un cliente ejecutando las peticiones por medio de las librerías libcoap para CoAP y cURL para HTTP desde Linux Ubuntu.

En el documento referenciado se analizan las características de bytes transmitidos, tiempo de respuesta y consumo de energía. Para los dos primeros se utiliza la herramienta de captura de mensajes de red Wireshark mientras que para hallar el consumo de energía que supone utilizar cada uno de los protocolos se realiza una simulación por medio de Cooja.

El objetivo principal de este TFG era reproducir la comparación entre protocolos presente en el artículo, pero utilizando un circuito físico simple en el entorno del Internet de las Cosas, evitando simular ninguna de las características y basando todos los resultados en datos reales. La variación del consumo de energía que supone utilizar un protocolo u otro en un mismo escenario es un dato importante a tener en cuenta y se ha querido obtener unos valores reales de los mismos.

Además del objetivo principal, se decidió añadir al análisis algunas funcionalidades del protocolo CoAP interesantes. Estas funcionalidades son:

- La diferencia en el intercambio de mensajes cuando se realiza de forma fiable y no fiable, es decir, cuando se utiliza un mensaje de tipo Confirmable en la petición y cuando se utiliza uno No Confirmable. De esta manera se podrá comprobar si el mismo protocolo ofrece unas prestaciones distintas durante el envío en función de la fiabilidad del intercambio.

- La característica de descubrir los recursos que gestiona un servidor utilizando la arquitectura CoRE que implementa el protocolo CoAP. Se trata de un mecanismo útil en el sector del Internet de las Cosas por su capacidad para autonomizar dispositivos y permitir una comunicación M2M sin necesidad de intervención humana.

- Utilizar la opción de "*observe*" [11] que permite observar un recurso en concreto que gestiona un servidor. Esta opción no existía en el planteamiento original del protocolo y se trata de una extensión del mismo. El sistema de intercambio de mensajes que utiliza esta opción es similar al método *publish/suscribe* descrito en la parte de MQTT, pero sin necesidad de utilizar un *broker* para ello, sino que este intercambio se gestiona directamente entre el cliente y el servidor.

4.2. Escenario

Los casos prácticos están formados por una serie de intercambios de mensajes entre un cliente y un servidor que contiene la información de la temperatura y humedad ambiente obtenida por medio de un sensor. El envío de los mensajes se realiza vía internet por medio del protocolo de estudio CoAP, el cual se pretende analizar, y el protocolo HTTP utilizado como referente. Se realiza una comparativa entre ambos protocolos para comprobar la veracidad de la teoría plasmada en el apartado que describe el funcionamiento del protocolo CoAP.

Para realizar las pruebas, se ha optado por utilizar una herramienta con muchas posibilidades y proyección en el mundo de las IoT. Esta herramienta es Arduino [12]. Arduino es una plataforma de hardware libre de código abierto basada en una placa con un microcontrolador y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares.

Se ha escogido Arduino por la facilidad para desarrollar elementos autónomos, la posibilidad de conectarse a dispositivos, e interactuar tanto con hardware como con software.

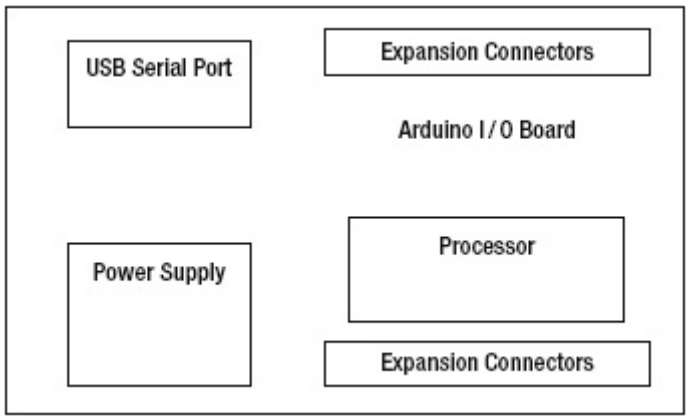

Figura 11: Diagrama de bloque de Arduino

Por un lado, la parte de hardware está basada en un microcontrolador, es decir, en un circuito integrado programable con la capacidad de ejecutar las instrucciones grabadas en su memoria. A su vez, este microcontrolador está compuesto por tres unidades funcionales que son: unidad central de procesamiento, más conocida como CPU, la memoria, y los periféricos de entrada y/o salida.

Arduino está constituido por tres tipos de memoria diferentes: la memoria Flash, la SRAM y la EEPROM. La primera es la correspondiente al disco duro, donde se guarda el *sketch* compilado y también el bootloader (encargado de ejecutar el Arduino en sí). La EEPROM es donde se guarda la información que se quiere que permanezca después de reiniciar el Arduino ya que, al igual que la memoria flash, se trata de una memoria no volátil. Por último, la memoria SRAM es volátil y es donde se crean y manipulan todas las variables en tiempo de ejecución.

Arduino cuenta con una amplia variedad de versiones de su propio Hardware adaptadas para los distintos tipos de requisitos que necesite el usuario. Dentro del conjunto de dispositivos que hay, se pueden distinguir dos grupos bien diferenciados, que son las placas y las shields:

- Las placas son lo que se conoce propiamente como “Arduino”, es decir, la base de cualquier programa que utilice esta herramienta.

- Las shields son extensiones compatibles con la base que se pueden colocar en la parte superior de los arduinos y permite comunicarse con ellos además de ampliar las funcionalidades de los mismos.

Por otro lado, el software consiste en un entorno de desarrollo integrado (IDE). Se trata de un programa informático formado por un editor de código, un compilador, un depurador, un constructor de interfaz gráfica (GUI) y las herramientas necesarias para cargar el código compilado en la memoria flash del hardware. Al existir una gran variedad de placas, el software incorpora una gestión de placas y librerías para poder adaptar las herramientas de desarrollo al entorno en el que se trabaja.

Los *sketches* hacen referencia al programa en sí, el lugar donde se edita el código de la aplicación y que posteriormente se compila y ejecuta en el Arduino. El lenguaje utilizado para programar es un lenguaje propio que está basado en C++, pero aplicado a microcontroladores. El *sketch* tiene dos elementos básicos que son obligatorios en todos los programas: setup y loop. Además, opcionalmente pueden incluirse librerías, definir variables y funciones propias, y/o utilizar funciones predefinidas para Arduino como *pinMode* (configurar el pin como entrada o salida), *digitalWrite* (cambiar el valor de salida de un pin) o *delay* (añadir un retardo) entre otros.

El setup corresponde a la configuración y se llama una única vez cuando el programa comienza. Generalmente en este punto se inicializan las librerías, se configura los distintos componentes conectados o se gestionan los pines que se van a utilizar. Por otro lado, el loop es el corazón del *sketch*, donde se ejecuta realmente el programa. Como su nombre indica, se trata de un proceso que se repite en bucle.

```
void setup() {
    initializeSerial(); // initialize Serial and Serial3 at 115200
    Baud rate
    setupESP8266(); // connect and configure the ESP-01
    dhtSensor.setup(DHT_DATA_PIN); // set the input pin
    unsigned long period = dhtSensor.getMinimumSamplingPeriod()*1000;
    Timer3.initialize(period); // initialize timer3, and set the
    minimum period
    Timer3.attachInterrupt(updateTime); // attaches function as a
    timer overflow interrupt
    Serial.println("ready");
}
void loop() {
    refreshDHTSensor(); // refresh every 'period' time
    listenRequest();
}
```

Figura 12: Bloques setup y loop del código en Arduino

En el escenario del proyecto, dentro del setup se realizan las configuraciones de los Serial, de la conexión a internet por medio de WiFi del ESP8266 y el modo de funcionamiento del mismo, se selecciona el pin que lee las lecturas del DHT y se ajusta el temporizador para usar la frecuencia mínima con la que se pueden realizar las lecturas.

El loop por su parte, se encarga de actualizar los valores del sensor y de escuchar si se ha recibido una petición sobre algún recurso, en cuyo caso se enviará una respuesta.

El escenario descrito es común a ambos programas, tanto para CoAP como para HTTP, pero cada uno de ellos tiene diferentes configuraciones y formas de proceder a la hora de enviar mensajes.

Una vez introducida la base sobre la que se ha desarrollado la aplicación, se procede a aclarar el circuito utilizado para las pruebas experimentales. Por medio de la herramienta Fritzing, una aplicación de código abierto del campo de la electrónica, se ha realizado un duplicado del circuito diseñado para los casos prácticos:

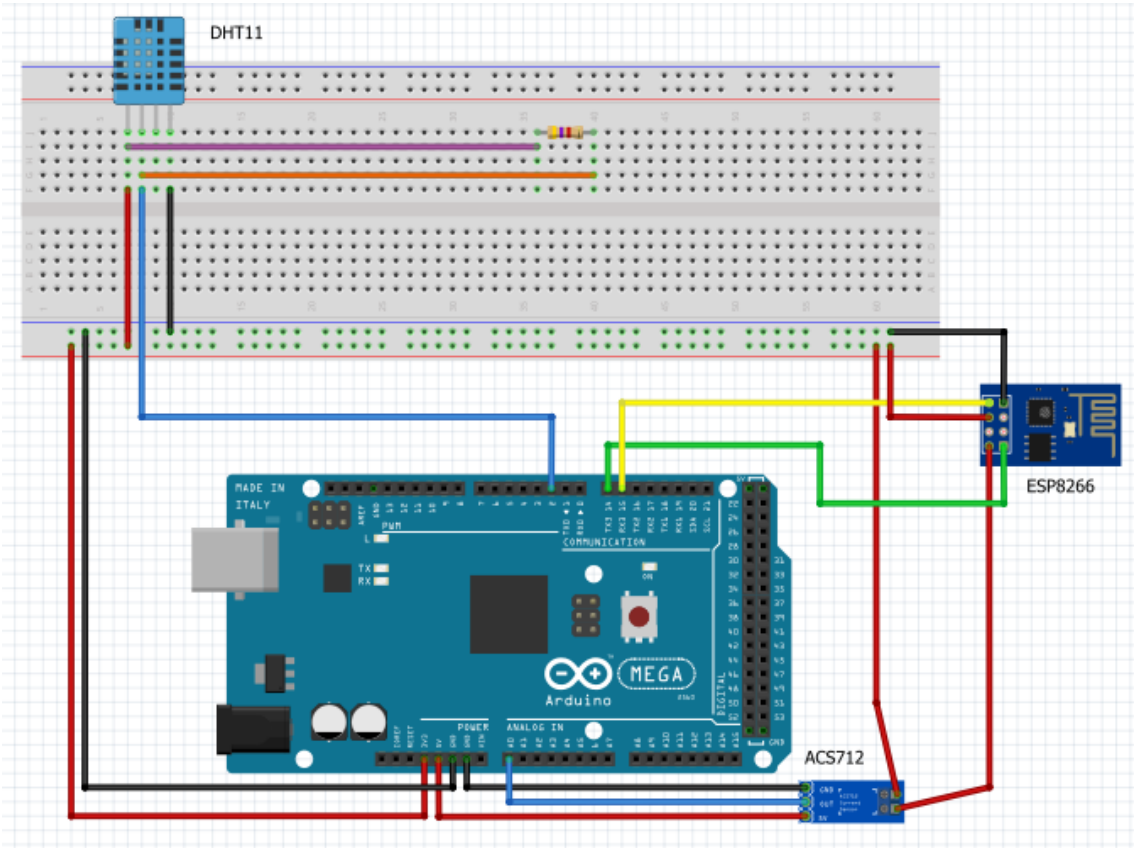

Figura 13: Circuito utilizado en los casos prácticos

En primera instancia se escogió un Arduino UNO para realizar los casos prácticos, sin embargo, tras montar el circuito y probar distintas versiones del código con la intención de adaptarlas a los requisitos del dispositivo, se comprobó que el mismo no tenía las capacidades de memoria suficientes para la extensión del programa.

En concreto, el problema tenía que ver con la memoria SRAM. A la hora de compilar, no se producían errores porque se cargaban sólo las variables globales, que no llegaban a ocupar todo el espacio, pero una vez se ejecutaba el programa comenzaba a fallar. El fallo se debía a que las variables locales no se tienen en cuenta durante la compilación, ya que se cargan una vez se ejecuta el programa. Se investigó en distintas guías [13] y foros [14] para reducir el uso de memoria SRAM, modificando la forma de declarar variables o liberando parte de la carga en la memoria Flash, pero aun así no era suficiente.

Así bien, finalmente se optó por utilizar un Arduino MEGA 2560 Rev3 que sí cumplía con las necesidades del proyecto con una memoria SRAM de 8 kB, es decir, el cuádruple que la de Arduino UNO con 2 kB.

Además de la placa, se han utilizado tres componentes que son compatibles con Arduino, cada uno de ellos con una función en el proyecto:

- DHT11 [15]: sensor de temperatura (de 0ºC a 60ºC con precisión de 1ºC) y humedad relativa ambiente (de 0% a 60% con precisión del 5%).

- ESP8266-01 [16]: microchip integrado con conexión WiFi y compatible con los distintos protocolos del modelo TCP/IP.

- ACS712-30A [17]: sensor de corriente con un rango de -30 A a 30 A.

La elección de los dos primeros componentes se ha producido por el hecho de simular una situación real en la que desde una sede de control se quisiera tener un seguimiento frecuente pero no obligatoriamente constante de las condiciones atmosféricas de un determinado terreno.

Al tratarse de pruebas realizadas siempre dentro de una sala, con unos valores de temperatura y humedad bastante estables dentro de un rango poco exigente, y que la exactitud de los datos no era imprescindible, no era necesario disponer de dispositivos más caros y con más precisión que los escogidos. Además, la idea principal de los casos prácticos era la de trabajar con un sistema de información sobre las condiciones atmosféricas conectado a internet que no fuese simulado, sino elaborar un caso real. Tanto el DHT11 como el ESP8266-01 cumplían a la perfección con el cometido que se les exigía, es decir, el primero debía aportar los datos ambientales aproximados y el segundo ofrecer conexión a internet y soporte a los distintos protocolos utilizados.

Así pues, una vez identificados todos los elementos, se continúa explicando el interconexionado. La fuente u origen de todo el circuito es el Arduino, el cual se encarga de gestionar el envío y recepción de las señales entre los distintos componentes que están conectados a él. La placa de Arduino utiliza como fuente de alimentación un cable USB tipo A/B, que al mismo tiempo sirve para cargar el programa dentro del dispositivo y comunicarse con él mediante comandos. El módulo WiFi ESP8266 acepta como máximo 3,6 V por lo que es necesario que se conecte al pin del Arduino que profiere 3,3 V. Este valor también permite el correcto funcionamiento del DHT11 ya que su rango de voltaje de operación es desde 3 V

hasta 5,5 V. Para alimentar ambos elementos se ha utilizado una protoboard y así poder compartir tanto la tensión como la tierra o GND.

El ACS712 por su parte necesita entre 4,5 V y 5,5 V de alimentación para funcionar correctamente por lo que, al ser el único componente que requiere este rango, se conecta directamente al pin de Arduino de 5 V. Como se quiere comprobar el consumo del ESP8266, la carga se obtiene de conectar el componente ACS712 entre la patilla de Vcc del módulo WiFi y la alimentación en la protoboard ya que la intensidad se debe medir en serie.

Si bien es verdad que se puede configurar muchos pines de la placa para adquirir la funcionalidad de comunicación bidireccional con distintos componentes, Arduino posee varios pines (el número difiere en función de la versión de Arduino que se utiliza) preparados para funcionar como transmisor y receptor de señales que trabajan en paralelo. En este caso en concreto, el Arduino Mega posee 3 pines con estas características y se ha optado por utilizar el "Serial" 3, que es como se denomina a este par de pines. Para ello, se ha conectado el pin de transmisión del Serial3 (o TX3) al pin de recepción del ESP8266 y el pin de recepción del Serial3 del Arduino (o RX3) al pin de transmisión del ESP8266. De esta forma, ambos dispositivos se comunican entre ellos pudiendo enviarse mensajes y respondiendo a los mismos.

Por último, para gestionar la conexión del sensor de temperatura se necesitaba realizar dos pasos. Por un lado, para la transmisión de datos entre el DHT11 y la placa de Arduino sólo se requería de un pin digital de la placa conectado a la segunda patilla del sensor. Por el otro, en la patilla anterior debía haber una resistencia de pull-up de 5kΩ ya que el cable utilizado era menor de 20 metros.

Esto es todo respecto a la gestión hardware del proyecto. En cuanto a la parte de software, se ha optado por desarrollar dos programas independientes en el IDE de Arduino, uno para el envío mediante CoAP y otro para HTTP. La base de ambos programas es común y la diferencia reside en la manera en que funciona el ESP8266 para cada protocolo.

Imitando el comportamiento de un sistema con sensores real, en ambos casos se optó por utilizar el módulo de WiFi ESP8266-01 como servidor de forma que el circuito estuviese realizando lecturas constantes por medio del sensor DHT11, pero sólo enviase la información cuando un cliente realizase una petición.

Para implementar el código de las aplicaciones en el entorno de desarrollo de Arduino IDE, se ha apoyado en varias librerías de terceros de código abierto que permiten el uso libre de su software.

Se pueden diferenciar las siguientes librerías comunes a ambos programas:

- TimerThree [18]: se trata de un temporizador que sirve para tomar valores cada "X" tiempo, establecido por el usuario, del sensor de temperatura y humedad.

- DHT [19]: se trata de una librería para leer los valores que devuelven los distintos sensores de temperatura y humedad de la familia DHT, en nuestro caso el DHT11.

- ESP8266 [20]: se trata de una librería que contiene los métodos para configurar y controlar los distintos dispositivos ESP8266.

En el caso de CoAP ha sido necesario añadir una cuarta librería, microcoap [21], para implementar las distintas funciones de este protocolo. Se trata de una librería diseñada para cargar un servidor de tipo CoAP en microcontroladores y está formada por los métodos para recibir las peticiones, desglosarlas en las cabeceras, opciones y carga útil, procesarlas y, por último, responderlas. Además, también están incluidas funciones del grupo de trabajo CoRE, que se nombró previamente en el apartado de CoAP, y permiten descubrir los recursos que gestiona el dispositivo, y la opción de añadir observadores a un recurso que controle el servidor.

Las desventajas encontradas en esta librería son que la misma sólo permite respuestas ACK de tipo *Piggybacked*, el conexionado y los dispositivos utilizados para el que estaba configurado no era el mismo que en este proyecto, y que basaba su conexión a internet en una shield de Arduino en lugar de un ESP8266. Por ello fue necesario realizar ciertos cambios en el código de la librería para adaptarla a los requisitos del escenario.

Una vez descrita la parte del servidor, se procede a mostrar el esquema completo del sistema de intercambio de mensajes (ver figura 14)

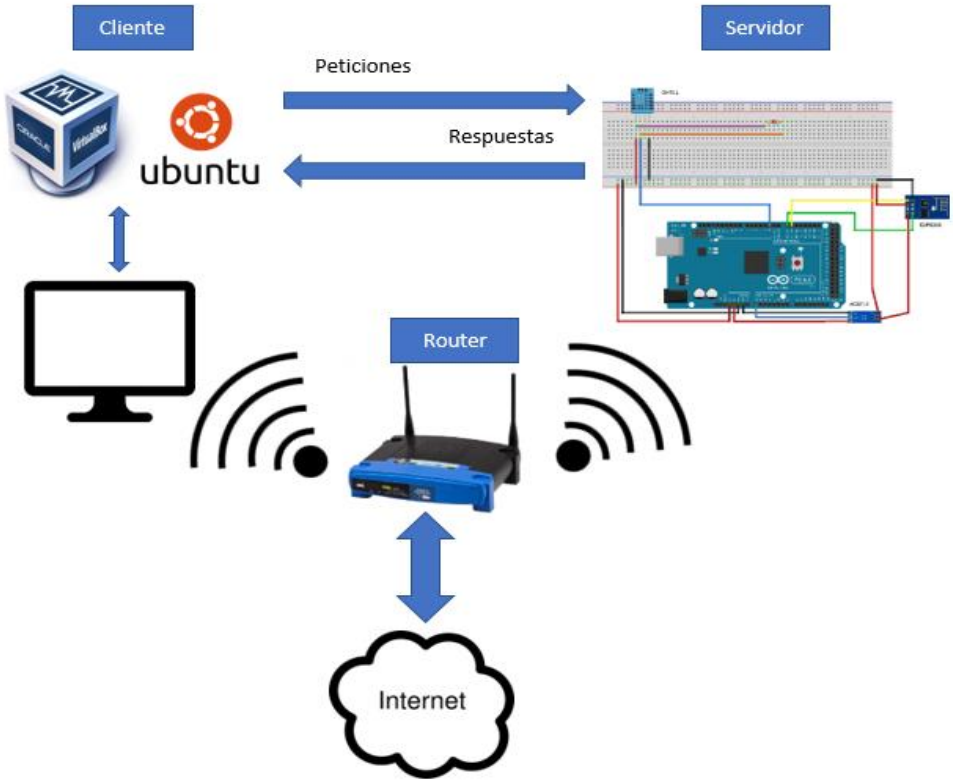

Figura 14: Esquema físico del intercambio de mensajes

Partiendo desde el lado del cliente, y con el objetivo de simular las peticiones, se usaron las librerías libcoap, para la parte de CoAP, y libcurl, para la parte de HTTP. El problema encontrado en este punto fue que la librería de CoAP requería de un sistema operativo de tipo Linux, Mac OSX u otras plataformas POSIX mientras que el ordenador del que se disponía sólo tenía el sistema operativo de Windows. Para solventar esta cuestión se optó por realizar las peticiones desde una máquina virtual. En este caso, la aplicación en cuestión era Oracle VirtualBox con la versión 16.04.3 LTS de Ubuntu, Linux.

Una diferencia a la hora de instalar entre las dos librerías es que la librería libcurl se obtiene directamente del sistema de administración de paquetes de Ubuntu por medio de un comando “apt-get install” mientras que libcoap es necesario descargarla manualmente desde el repositorio de internet.

A modo de resumen del esquema completo correspondiente al intercambio de mensajes, se define la siguiente tabla:

Tabla 1: Esquema de los componentes utilizados para el intercambio

Rol	Componente	Descripción
Cliente	PC	Provee los medios para ejecutar la máquina virtual
	Máquina virtual	Contiene el Sistema Operativo Ubuntu Linux
	Librería libcurl	Realiza las labores de cliente del protocolo HTTP
	Librería libcoap	Realiza las labores de cliente del protocolo CoAP
Servidor	PC	Fuente de alimentación, editor y compilador del programa de Arduino
	Arduino	Comunica los componentes del circuito y procesa la información de los mismos
	Sensor DHT 11	Obtiene los datos de temperatura y humedad ambiente
	Módulo WiFi ESP8266	Provee conexión a internet y los protocolos necesarios para el intercambio de mensajes
	Sensor ACS712	Obtiene los datos de intensidad que pasa por el ESP8266

Otra opción era usar la extensión “*Copper*” que ofrece el navegador Firefox. Esta extensión permite ejercer la labor de cliente de CoAP y mandar peticiones a un servidor. Esta opción se descartó por no ser compatible para versiones más actualizadas del navegador además de no tener un gran soporte.

Una vez preparado el entorno se pudo proceder a comprobar ambos programas, capturando el tráfico con la aplicación Wireshark, que es una herramienta que sirve para analizar el flujo de mensajes que se envían a través de una red y que permite distinguir entre los distintos protocolos existentes y filtrarlos.

4.3. Casos prácticos

4.3.1. Caso práctico 1: Petición y Respuesta en CoAP y HTTP

En el primer caso, la finalidad del experimento es comparar las características técnicas de los protocolos CoAP y HTTP en un intercambio lo más sencillo posible. Para ello, en ambos protocolos se envía una petición de tipo GET solicitando la información correspondiente al recurso de la temperatura. Se realiza una serie de 20 peticiones consecutivas para poder analizar a fondo el número de bytes transmitidos, el tiempo de respuesta y el consumo de energía. Las peticiones de CoAP son de tipo Confirmable y las respuestas son de tipo ACK.

Las queries utilizadas para los dos protocolos son:

HTTP: *curl -G http://192.168.1.140/dht_sensor/temperature*
CoAP: *./coap-client -m get coap://192.168.1.140/dht_sensor/temperature*

Como se puede observar en las queries de petición, el servidor posee la dirección IPv4 192.168.1.140. Lo ideal hubiera sido utilizar una dirección IPv6, pero el componente ESP8266 no posee esta característica.

Se realiza un análisis de las características nombradas anteriormente con los resultados obtenidos en las distintas peticiones:

- Número de bytes transmitidos: se trata de la cantidad de información intercambiada entre el cliente y el servidor. En ambos protocolos, independientemente del número de repeticiones y de los valores de temperatura obtenidos, el tamaño de las peticiones y las respuestas era siempre el mismo. Para los mensajes generados en este caso práctico se han obtenido los siguientes datos:

Tabla 2: Tamaño de los mensajes intercambiados en el primer caso

Protocolo	Tamaño petición (bytes)	Tamaño respuesta (bytes)
HTTP	131	156
CoAP	69	56

Como en el caso de estudio la carga útil es similar en ambas variantes, es razonable asumir que las diferencias residen en el tamaño de las cabeceras de cada uno de los protocolos y en la capa de transporte que utilizan para este intercambio. Para analizar esta teoría, se ha recurrido a la herramienta "*Wireshark*" que permite la captura y desglose de los mensajes.

Los datos que se visualizan en la tabla corresponden al mensaje completo, por ello, se han desglosado los mensajes en las diferentes capas que los conforman. Tras esta segregación se ha podido observar que, para ambos protocolos, los primeros niveles correspondientes a la capa física, la capa de enlace de datos y la capa de red, están formados por la misma cantidad de bytes.

La primera diferencia se contempla en la capa de transporte, mientras que HTTP se transmite por TCP con una cabecera de 20 bytes sobre esta capa, en CoAP se transmite sobre UDP cuya cabecera tiene 8 bytes. El tamaño restante de los mensajes corresponde a los protocolos de estudio en sí. La capa de aplicación referente a HTTP es de 57 bytes para la petición y 82 bytes para la respuesta y la de CoAP es de 27 bytes para la petición y 16 bytes para la respuesta.

Tras la separación por capas se ha podido comprobar que efectivamente el tamaño de HTTP es muy superior al de CoAP, lo cual es una desventaja para los dispositivos con una escasa capacidad de recursos.

- Tiempo de respuesta: se denomina tiempo de respuesta al intervalo de tiempo que tarda un mensaje en ser contestado. La medida exacta se obtiene calculando el tiempo que pasa desde el instante en el que el cliente envía el mensaje con la petición hasta que recibe el mensaje con la información solicitada. Los valores obtenidos para ambos protocolos en 10 repeticiones independientes se muestran en una gráfica (ver figura 15).

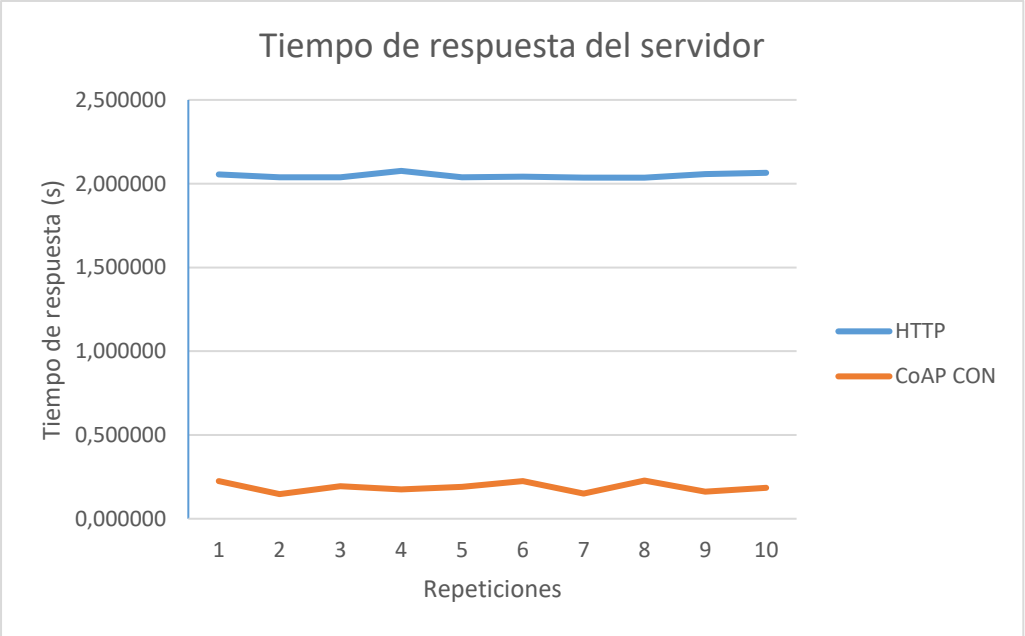

Figura 15: Gráfica del tiempo de respuesta de los servidores del primer caso

Como se puede observar en la gráfica, CoAP tiene un tiempo de respuesta medio de 190 milisegundos mientras que el de HTTP es superior a los 2 segundos. Estos datos demuestran que para situaciones como la que se ha

realizado en el experimento, es decir, en casos de transmisión directa entre dos dispositivos bajo la misma red, HTTP es prácticamente 10 veces más lento que CoAP.

Esta diferencia se ve incrementada debido a que HTTP se transmite mediante TCP, que tiene un carácter fiable, ya que garantiza la entrega de los paquetes, pero es más lento por los procesos de conexión, desconexión y control de paquetes que utiliza para asegurar esa fiabilidad. CoAP por su parte se transmite por medio de UDP, un protocolo que ofrece los servicios inversos, es decir, un protocolo poco fiable por la ausencia de garantías en el envío de mensajes, pero rápido por la escasa carga de requisitos.

En aplicaciones con sensores es importante mantener un tiempo de respuesta rápido, ya que la cantidad de datos a enviar es escasa y el tiempo de espera para obtener los mismos debería ser proporcional a su tamaño.

- Consumo de energía: hace referencia a los requisitos necesarios para poner en funcionamiento cada uno de los protocolos estudiados. Para obtener esta característica se ha partido de los valores obtenidos por el sensor de corriente ACS712.

  Se ha elegido el pin analógico A0 para realizar la lectura de los valores de la intensidad. Este componente transforma la lectura de la intensidad en una tensión proporcional dentro del rango de actuación del mismo. Como se comentó previamente, Arduino posee varias funciones predefinidas y en este caso se utiliza “analogRead” para cargar los datos del componente. Esta función devuelve valores entre 0 y 1023, correspondientes a 0 V y 5 V respectivamente, por lo que es necesario realizar la siguiente transformación para obtener la intensidad:

$$I = \frac{V - 2.5}{Sensibilidad}$$

  Donde I es intensidad, V el voltaje que se devuelve la función analogRead de la salida del ACS712, y la Sensibilidad para el componente en cuestión viene dada por el fabricante, con un valor de 66 mV/A.

  Para este caso se ha realizado una serie de 20 peticiones para cada protocolo. En cada una de ellas se ha realizado una lectura después de recibir la petición y otra después de enviar la respuesta. Estas lecturas realmente corresponden a la media de 200 instantes consecutivos, pero se devuelve un único valor para tener una salida más fiable. Dentro del código se han incluido los cálculos pertinentes para obtener por salida directamente el valor de la intensidad. Se ha realizado una gráfica con las diferentes lecturas obtenidas para ambos protocolos (ver figura 16).

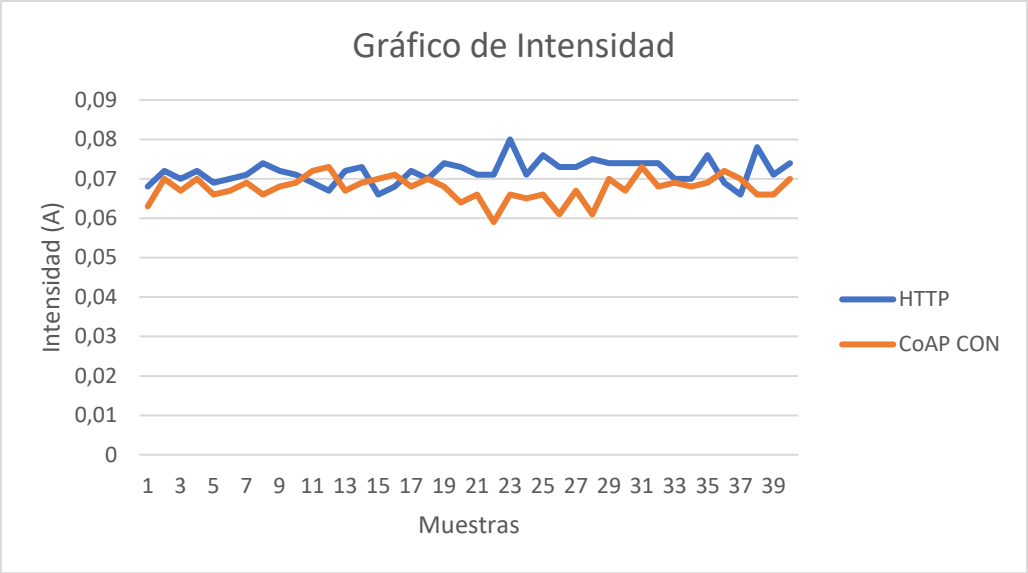

Figura 16: Gráfica de las lecturas de la intensidad del primer caso

La intensidad media obtenida es de 72 mA para HTTP y 68 mA para CoAP. La potencia consumida se consigue del producto de la intensidad media obtenida por la tensión. Aunque teóricamente Arduino suministra una alimentación de 3,3 V se ha preferido comprobar con un multímetro la tensión que llega al pin Vcc del ESP8266 para tener el valor real. El voltaje obtenido ha sido de 3,2 V por lo que la potencia media es:

$$\text{HTTP: } P = I * V = 0{,}230W$$
$$\text{CoAP: } P = I * V = 0{,}218W$$

Se concluye que el consumo es ligeramente inferior en el caso de CoAP, un resultado esperado teniendo en cuenta que el tamaño de los paquetes de HTTP es superior a los de CoAP y que TCP es un protocolo más pesado a la hora de procesar que UDP.

4.3.2. Caso práctico 2: Petición y Respuesta en CoAP de tipo No Confirmable

En este segundo caso, el propósito es ver cómo se comporta el protocolo CoAP al enviar una petición de tipo No Confirmable y comparar los resultados con los de los mensajes Confirmables de CoAP obtenidos en el caso práctico anterior. Para lograr el intercambio de CoAP con el tipo de mensaje No Confirmable, se ha partido de la query original del caso práctico 1 pero añadiendo el texto "-N" que indica la librería libcoap para que el mensaje se envíe de manera No Confirmable. La query resultante de la petición es la siguiente:

./coap-client -m get coap://192.168.1.140/dht_sensor/temperature -N

Los mensajes, tanto en la petición como en la respuesta, son de tipo No Confirmable por lo que durante el intercambio no se asegura la llegada de los mismos. A raíz de la repetición de peticiones, se ha podido comprobar cómo se ha dado lugar a pérdidas de paquetes, tanto en las solicitudes como en las respuestas.

Las características obtenidas en este caso práctico son las siguientes:

- El tamaño de las solicitudes y respuestas es idéntico al de los mensajes de tipo Confirmable y ACK respectivamente. Tiene sentido esta casuística puesto que el único cambio en la estructura del mensaje es en el campo del tipo de mensaje dentro de la cabecera que no afecta al tamaño.

- La potencia consumida en el caso de los mensajes No Confirmables también tiene valores semejantes a los de tipo Confirmable dado que la gráfica y la media de la intensidad es similar en ambos casos. La intensidad media es de 0.0679 A por lo que podemos considerar el consumo de potencia medio igual para ambos casos. La intensidad media se ha obtenido de 20 medidas, al igual que en el caso práctico anterior, y se ha comparado los datos obtenidos con los de los mensajes Confirmables (ver figura 17).

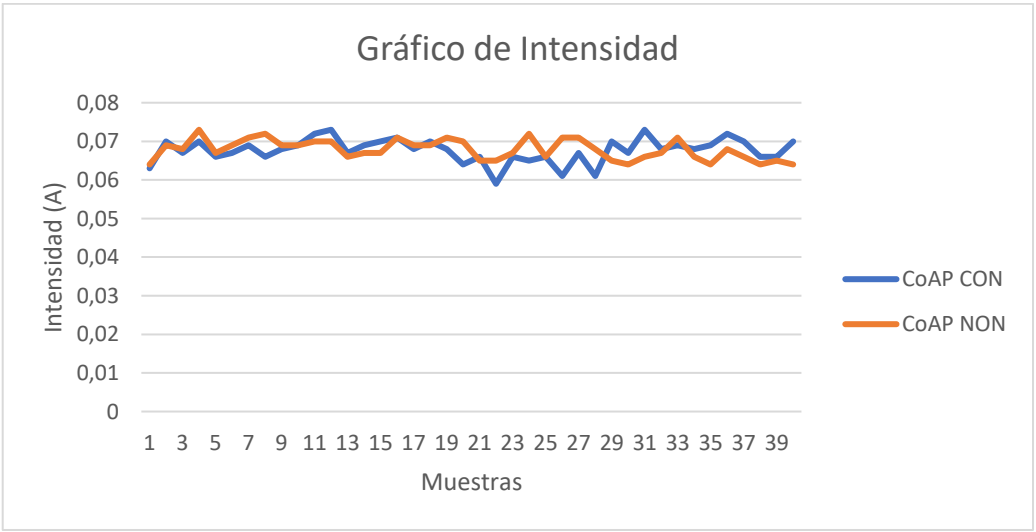

Figura 17: Gráfica de las lecturas de la intensidad del segundo caso

- Por último, en el tiempo de respuesta del servidor, al contrario que en las características anteriores, sí que se observa una diferencia frente a la obtenida en los mensajes Confirmables. El tiempo medio de respuesta del servidor es de 160 milisegundos en los mensajes NON frente a los 190 de los mensajes CON. Se genera una gráfica comparando los datos obtenidos con el caso anterior (ver figura 18).

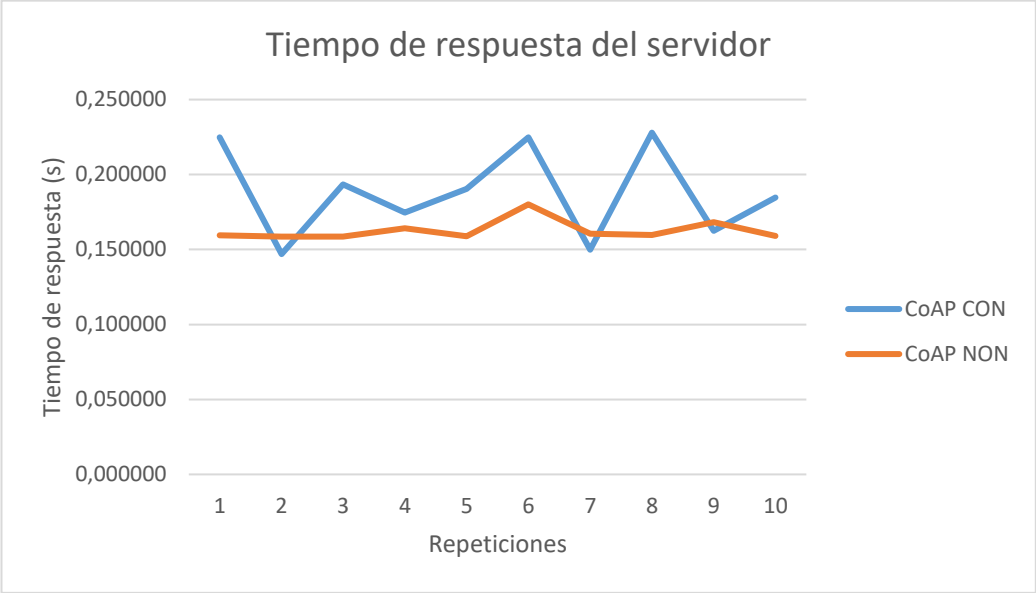

Figura 18: Gráfica del tiempo de respuesta de los servidores del segundo caso

Se concluye por tanto que los mensajes de tipo No Confirmable sacrifican la seguridad de recepción de mensajes que aportan los mensajes Confirmables a cambio de una respuesta más rápida por parte del servidor.

4.3.3. Caso práctico 3: Descubrir recursos de un dispositivo por medio de CoRE

En el tercer caso práctico se pretende comprobar el comportamiento de CoAP cuando se realiza una petición desde el cliente para descubrir los recursos que gestiona el dispositivo servidor por medio de CoRE.

Para ello, como se comentó en la definición de CoRE en la introducción a CoAP, en lugar de realizar la petición sobre un recurso en concreto como en los casos anteriores, se debe emplear el texto *"./well-known/core*" a cambio del "*dht_sensor/temperature*" utilizado previamente. De esta manera, la query enviada desde el cliente queda de la siguiente manera:

./coap-client -m get coap://192.168.1.140/.well-known/core

En este caso también existe la opción de utilizar mensajes de tipo No Confirmable para realizar la petición, aunque el propósito de estos mensajes es recibir la información de la estructura una única vez y correctamente para saber los recursos que gestiona el servidor, por lo que no tiene sentido utilizar esta vertiente.

Una vez obtenida la petición, el servidor contesta con un mensaje indicando la estructura de los recursos que gestiona. En ese caso, la respuesta es la siguiente:

</dht_sensor>;ct=40,</dht_sensor/temperature>;rt="temperature-c",</dht_sensor/humidity>;rt="humidity"

Esta respuesta indica que el dispositivo es un sensor DHT que gestiona los recursos de la temperatura en grados centígrados y de la humedad. A partir de aquí, un cliente podría seleccionar la nueva query para obtener los valores deseados. Este tipo de peticiones es muy útil cuando se desconoce el dispositivo al que se quiere preguntar y así poder obtener la información necesaria para realizar una petición sobre un recurso en concreto correctamente.

Las características obtenidas de este intercambio son las siguientes:

- El tamaño de la solicitud en este caso es ligeramente inferior, de 63 bytes, lo cual tiene sentido porque la query contiene menos caracteres que los casos anteriores. Ahora bien, la respuesta por su parte tiene un tamaño muy superior a lo que se ha visto hasta ahora, con 165 bytes. Es lógico que se observe esta diferencia cuando las respuestas anteriores sólo incluían el valor concreto de un recurso y en este caso se maneja una cantidad de texto muy superior. También hay que tener en cuenta que este tipo de respuestas se darán con poca frecuencia, puesto que su propósito es de utilizarlas una única vez para descubrir la estructura del servidor.

- El tiempo de respuesta medio es ligeramente inferior al del primer caso práctico, siendo el valor medio de 180 milisegundos, un resultado esperado

cuando se trata de un mensaje que también es de tipo Confirmable y además no tiene que cargar los datos del sensor, sino que simplemente responde con la información estándar preparada. Se compara los tiempos de respuesta con los del primer caso por medio de una gráfica (ver figura 19).

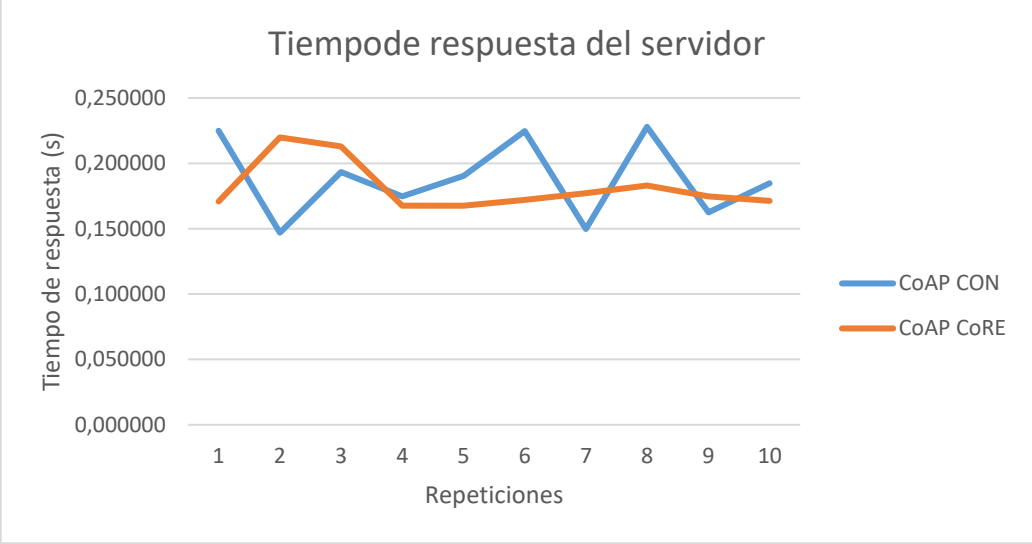

Figura 19: Gráfico del tiempo de respuesta de los servidores del tercer caso

- Los datos obtenidos de la intensidad en este caso son ligeramente superiores a los del primer caso práctico. El valor medio de los mismos se acerca a la intensidad que se obtenía en el caso de HTTP, aunque sin llegar a igualarlo, siendo este de 71 mA. Este valor da lugar a un consumo de potencia de 0.227 W. La intensidad en CoRE se compara con el primer caso práctico por medio de una gráfica (ver figura 20).

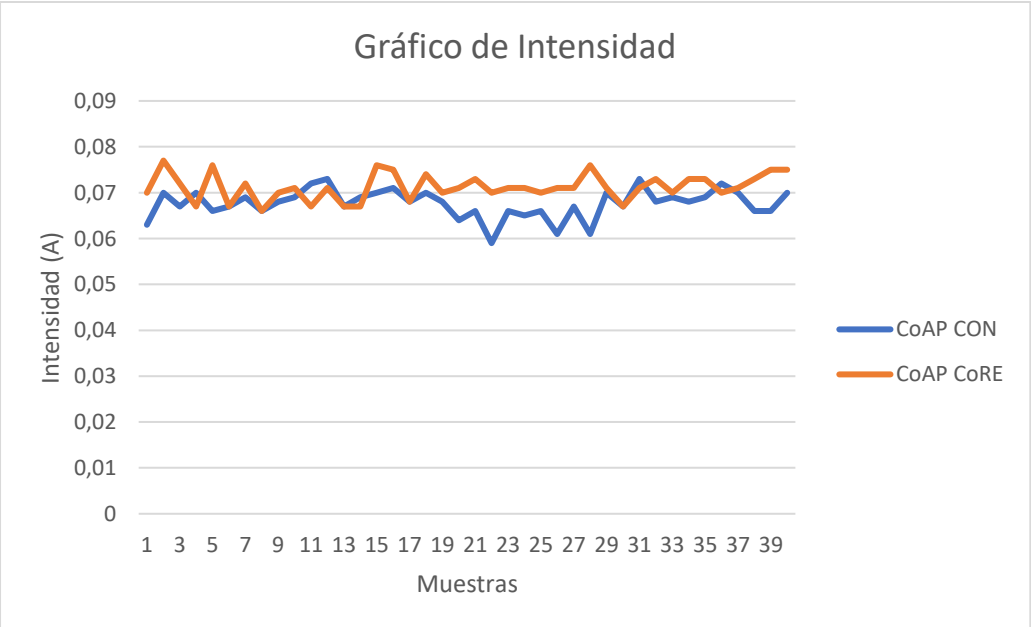

Figura 20: Gráfico de intensidad del tercer caso

La conclusión de este caso práctico es que, aunque los parámetros obtenidos en este tipo de intercambio sean peores, se trata de un mensaje útil que facilita el uso del IoT. El alcance de este tipo de mensajes es muy amplio, pudiendo llegar a automatizar, gracias al formato que poseen, los intercambios entre dispositivos para que se gestionen de forma autónoma sin acción humana de por medio.

4.3.4. Caso práctico 4: Observar un recurso por medio de CoAP

En el último caso se busca tener los datos de los sensores actualizados constantemente. Para ello, se ha hecho uso de la opción "*observe*" de CoAP, la cual permite obtener información sobre un recurso de un dispositivo cada cierto tiempo sin necesidad de mandar peticiones periódicas.

El servidor, después de recibir la petición, actualiza la lista de observadores añadiendo al cliente para obtener el recurso deseado. Una vez que el cliente termine de interesarse por el recurso observado, puede cancelar la observación del mismo, eliminando el servidor en ese momento a dicho cliente de la lista de observadores.

Para lograr esta opción, la librería libcoap indica que se debe añadir el texto "*-s*" a la query original seguido por la duración (en segundos) durante la que se desea observar el recurso. Para este caso, se indicó una duración de un minuto quedando la query de la siguiente forma:

./coap-client -m get coap://192.168.1.140/dht_sensor/temperature -s 60

El mecanismo seguido por el servidor, tras recibir la petición y añadir al cliente en la lista de observadores, es de enviar un mensaje de tipo No Confirmable cada vez que se actualiza el valor de la temperatura en la aplicación. Este envío se produce de manera periódica hasta que el cliente cancela la suscripción.

El intercambio de mensajes observado en este caso es el siguiente:

- En primer lugar, el cliente indica el recurso que quiere observar mediante una petición normal con un mensaje de tipo Confirmable, con la diferencia de que incluye una opción adicional frente al mensaje estándar. Desde la herramienta de Wireshark se puede observar la estructura de las opciones del mensaje y cómo incluye la opción Observe con valor 0. Una vez vencido el tiempo establecido en la query de suscripción, el cliente vuelve a enviar un mensaje de tipo Confirmable igual, pero en este caso la opción de Observe tiene el valor 1 para indicar que se cancela la suscripción. La estructura de las opciones de ambos casos se puede observar en la tabla inferior.

Tabla 3: Valores de las opciones para suscribir y cancelar la suscripción

Opción	Suscripción	Cancelación de suscripción
1. Observe	0	1
2. Uri-Path	dht_sensor	dht_sensor
3. Uri-Path	temperature	temperature

Además de los valores de Observe, se puede comprobar que la opción Uri-path se repite. Esto se debe a que en CoAP, cada vez que se utiliza el carácter "/"

para escoger un determinado recurso del servidor, la cadena de texto se representa como una opción Uri-path con el valor del texto.

- La primera respuesta por parte del servidor para los mensajes de suscripción definidos previamente es de tipo ACK, y contiene la información relativa al recurso incluida en esta respuesta. Las respuestas posteriores al ACK de suscripción son de tipo No Confirmable y tienen el mismo tamaño que el ACK puesto que la única diferencia entre ambos es que cambia el tipo de mensaje en la cabecera.

Las características obtenidas para este tipo de intercambios son que, por un lado, el tamaño de los mensajes es ligeramente superior al intercambio sin la opción de observar. La diferencia es de tan sólo 2 bytes, quedando la petición en 71 bytes y las diferentes respuestas en 58 bytes. Estos 2 bytes corresponden al tamaño de la opción Observe. Además de las peticiones, las respuestas también contienen esta opción, pero en lugar de utilizar los valores 0 y 1 para observar y dejar de observar un recurso, utilizan valores que se van sumando consecutivamente para que el cliente pueda detectar el orden en el que se envían los mensajes.

Por otro lado, el tiempo de respuesta del servidor cuando este incluye la opción de Observe es bastante superior al de un mensaje Confirmable normal, siendo aproximadamente de 300 milisegundos. Este tiempo de respuesta sólo se da en dos ocasiones, cuando se solicita y cancela la observación de un recurso. El resto del tiempo no hay una respuesta explícita, sino que hay un flujo unidireccional de mensajes desde el servidor hacia el cliente.

Por último, en cuanto al consumo de potencia, este es muy similar a los casos de intercambio estándar de mensajes Confirmables y No Confirmables.

En resumen, en estos casos se sacrifica el tiempo de respuesta a la hora de suscribirse o cancelar la suscripción, y una ligera carga superior de bytes en el intercambio de mensajes completo, a cambio de tener información actualizada en todo momento sin necesidad de realizar peticiones constantemente y saturar de esta manera el canal.

4.4. Resultados obtenidos

Para concluir con la parte práctica del proyecto, se expone un resumen de todos los resultados obtenidos en los casos prácticos y se compara con los datos aportados en la evaluación del artículo de referencia.

En primer lugar, se tiene en cuenta la característica de los bytes totales transmitidos durante un intercambio completo, es decir, teniendo en cuenta tanto la petición como la respuesta. En el caso del protocolo CoAP, la cantidad de bytes trasmitidos en el experimento varía en función del caso práctico, desde 125 en un intercambio estándar hasta 228 en el caso de *discover*, mientras que en el artículo el intercambio es de 184 bytes.

Por otro lado, en el caso del protocolo HTTP sí que se observa una gran diferencia, obteniendo un valor de 287 bytes para la transmisión completa en el caso práctico mientras que en el artículo el intercambio está formado por 1288 bytes.

Una diferencia observada es que en el caso práctico la información devuelta por el servidor es únicamente el valor del recurso solicitado mientras que en el artículo las respuestas contienen la dirección IP del sensor, dato que podrían suprimirse perfectamente, y los valores de temperatura y humedad. La disparidad en los intercambios no se puede atribuir al protocolo en sí, ya que depende directamente del desarrollo que se haya realizado en la aplicación.

En segundo lugar, el tiempo de respuesta del servidor es casi idéntico en ambos casos para el protocolo CoAP, con unos valores de 184 milisegundos en los datos del artículo y 190 milisegundos en el caso práctico. Aunque en el artículo no se indica explícitamente qué tipo de mensajes intervienen en el intercambio, se puede deducir por la query utilizada en la petición que el intercambio es el mismo que el utilizado en el primer caso práctico, es decir, un mensaje Confirmable para la petición y un ACK para la respuesta, de ahí que se haya comparado el valor de ese caso. En cuanto al tiempo de respuesta del servidor HTTP, el valor es ligeramente inferior en los resultados del artículo, de 1.8 segundos, frente a los 2 segundos del caso práctico.

Por último, el consumo de energía sí que supone una variación importante entre las dos opciones. En el artículo se hace un desglose de las distintas tareas que realiza el servidor (recepción, transmisión, procesado y modo de baja potencia) y se obtiene el consumo de energía en cada uno de ellos. En los casos prácticos no ha sido posible realizar un desglose en función de las tareas ya que no se poseía las herramientas necesarias para ello. Para lograr un análisis en profundidad se necesitaría un osciloscopio que permitiese observar la variación de la intensidad en función de las labores desempeñadas.

En su lugar, se ha optado por obtener el consumo medio de potencia durante todo el intercambio de mensajes, lo que incluye la recepción, procesado y transmisión de estos. Para todos los tipos de intercambio en CoAP los valores de potencia consumida son inferiores a los que ofrece el protocolo CoAP.

5. MARCO REGULADOR

5.1. Normativa relativa al Internet de las Cosas

Este apartado se centra en las medidas que se han tomado a lo largo de los años y que afectan directa o indirectamente al ámbito del Internet de las Cosas en general. Para ello, se han tenido en cuenta las normativas europeas, que son las que marcan el camino que se debe seguir en España respecto a los temas nombrados previamente.

Los primeros avances que se tomaron en torno a dispositivos intercomunicados tuvieron lugar en 2005 cuando la Comisión Europea estableció la ruta a seguir en los años venideros. Este marco estratégico se denominó "*i2010: la sociedad de la información y los medios de comunicación al servicio del crecimiento y el empleo*" [22] y en él se especificaron las metas que se debían lograr antes del año 2010: la consecución de un espacio europeo único de la información; el impulso de la innovación y de la inversión en el campo de la investigación en las tecnologías de la información y la comunicación (TIC), y la consecución de una sociedad de la información y los medios de comunicación basada en la inclusión.

En abril de 2014 se propuso la Directiva 2014/53/UE relativa a la armonización de las legislaciones de los Estados miembros sobre la comercialización de equipos radioeléctricos [23]. En ella se definen las características que deben satisfacer los dispositivos radioeléctricos en el mercado, tales como la protección de la salud y seguridad físicas, los datos personales y la privacidad en internet, el correcto uso del espectro radioeléctrico o la interoperabilidad de los mismos.

En marzo de 2015 se creó la Alianza para la Innovación del IoT (AIOTI) [24] cuyo propósito es generar un flujo de diálogo entre las empresas europeas interesadas en el sector del IoT. Este grupo fomenta la innovación de los distintos sectores por medio de la transformación de modelos de negocio adaptados a los tiempos venideros.

En abril de 2016 la Comisión Europea publicó un documento de trabajo del personal de la comisión sobre los avances del Internet de las Cosas en Europa [25]. Este documento indica que los pilares principales para el desarrollo del IoT deben ser:

- Tener un mercado único para el IoT, es decir, que los dispositivos y servicios sean válidos en cualquier país de la Unión

- Un ecosistema próspero para IoT en el que se tienda a utilizar plataformas abiertas para ayudar a las comunidades de desarrolladores a innovar.

- Que el IoT esté centrado en el ser humano, poniendo siempre por delante la seguridad y la protección de datos.

Con respecto al caso que se simula en este trabajo, es importante nombrar la iniciativa *de "Comunidades Conectadas"* [26] puesto que esta intenta promover el despliegue de banda ancha de alta velocidad en las zonas rurales. El objetivo de esta iniciativa es el

de poner en contacto a los responsables de las localidades, las asociaciones locales de banda ancha y a los operadores, con asesores que puedan aconsejarlos sobre formas de financiación y poder crear modelos de empresa para llevar la banda ancha a esas regiones. Estas medidas fomentan el uso de sistemas IoT como el de sensores que se ha nombrado.

5.2. Estándares técnicos

El IETF (Grupo de Trabajo de Ingeniería de Internet) es la organización internacional encargada de la normalización y correcto funcionamiento de la arquitectura y los protocolos de Internet. Es una entidad abierta para que cualquier persona pueda publicar propuestas con su respectiva coherencia y, si estas pasan una serie de requisitos, pueden llegar a convertirse en un estándar.

Estas propuestas de los usuarios son conocidas como RFC (Peticiones de Comentarios). Las RFC se caracterizan porque tienen un título y número únicos que no pueden repetirse ni ser eliminados.

Respecto al presente trabajo, dentro del marco de desarrollo de la parte práctica, se han tenido en cuenta los siguientes estándares:

- En primer lugar, hay que distinguir el entorno sobre el que se ha desarrollado la aplicación. Por ello se ha de tener en cuenta la "*RFC 7228 - Terminology for Constrained-Node Networks*" [27], la cual incluye la definición de los nodos y las redes limitadas y una segregación de las mismas en función de sus características.

- El formato de los mensajes de CoAP se basan en el estándar del grupo de trabajo CoRE para permitir el descubrimiento de los recursos que gestiona cada uno de los dispositivos y se define en la "*RFC 6690 - Constrained RESTful Environments (CoRE) Link Format* ".

- En el caso del protocolo CoAP en sí, el RFC original que define el protocolo y el funcionamiento del mismo, y en el que se ha basado principalmente en este proyecto es el estándar "*RFC 7252 - The Constrained Application Protocol (CoAP)".*

- Además de la definición del protocolo CoAP, han ido surgiendo extensiones sobre la forma en la que se define el protocolo o la manera en la que funciona el mismo. Así, se pueden distinguir las siguientes propuestas:

    a. *RFC 7390 - Group Communication for the Constrained Application Protocol (CoAP)* [28]: se trata de una propuesta experimental para evaluar el uso de CoAP sobre grupos de dispositivos. Si bien es verdad que en el proyecto sólo se disponía de un dispositivo, es importante tener en cuenta esta evaluación por la proyección de futuro que posee.

b. *RFC 7641 - Observing Resources in the Constrained Application Protocol (CoAP)*: se trata del estándar utilizado en el cuarto caso práctico y hace referencia a una extensión de CoAP que permite "observar" los recursos, es decir, de obtener una representación de los mismos y mantenerla actualizada durante un periodo de tiempo. Se ha basado en este estándar para el caso práctico 4 de este proyecto.

c. *RFC 7959 - Block-Wise Transfers in the Constrained Application Protocol (CoAP)* [29]: se trata de un estándar que establece la forma de transmitir mensajes por medio de bloques cuando la carga útil es demasiado grande y requiere fragmentación de los paquetes. Este estándar no se ha tenido en cuenta en el proyecto ya que los mensajes intercambiados no tenían un tamaño lo suficientemente grande como para tener que fragmentarse.

d. *RFC 8323 - CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets* [30]: se trata de una propuesta que define los cambios necesarios en el formato de los mensajes de CoAP para permitir su transporte por medio de TCP, TLS y WebSockets. Además, engloba las transformaciones pertinentes para incluir en este tipo de envíos los RFC de CoAP definidos previamente.

6. ENTORNO SOCIO-ECONÓMICO

6.1. Presupuesto de la elaboración del TFG

El presupuesto de la elaboración del TFG se divide en dos áreas principales. La primera concierne a todos los elementos utilizados para la realización del caso práctico y la segunda corresponde con el valor humano, en función de las horas dedicadas en todo lo que atañe al trabajo.

Para la primera parte se ha contado con los siguientes componentes:

Tabla 4: Componentes Utilizados

Componentes	Precio (€)
Ordenador Portátil HP dv6-6b13ss	1100
Arduino UNO rev3	20
Arduino MEGA 2560 rev3	35
Módulo WiFi ESP8266-01	7.5
Multímetro PS-Tech MB01	12
Sensor de corriente ACS712-30A	5.81
Sensor de temperatura y humedad DHT11	3.3
Placa protoboard	7.8
Cables DuPont macho-macho, macho-hembra, hembra-hembra	3.4
Resistencias	0.5
Gastos de envío totales	10.2
Total	**1205.51**

Para la segunda parte hay que hacer un desglose de las horas dedicadas a cada función del trabajo, que se pueden dividir en las siguientes:

- Investigación: en este punto se incluye el tiempo dedicado a la obtención de conocimientos.

- Desarrollo: está constituido tanto por el diseño y elaboración del circuito como por la programación de las aplicaciones utilizadas en el caso práctico.

- Elaboración de la memoria: como el nombre indica, está compuesto por el esfuerzo dedicado en la redacción de la memoria.

Apoyado en el desglose de tareas, se ha realizado un diagrama de Gantt que contiene en detalle la dedicación a cada uno de los cometidos anteriores (ver figura 21).

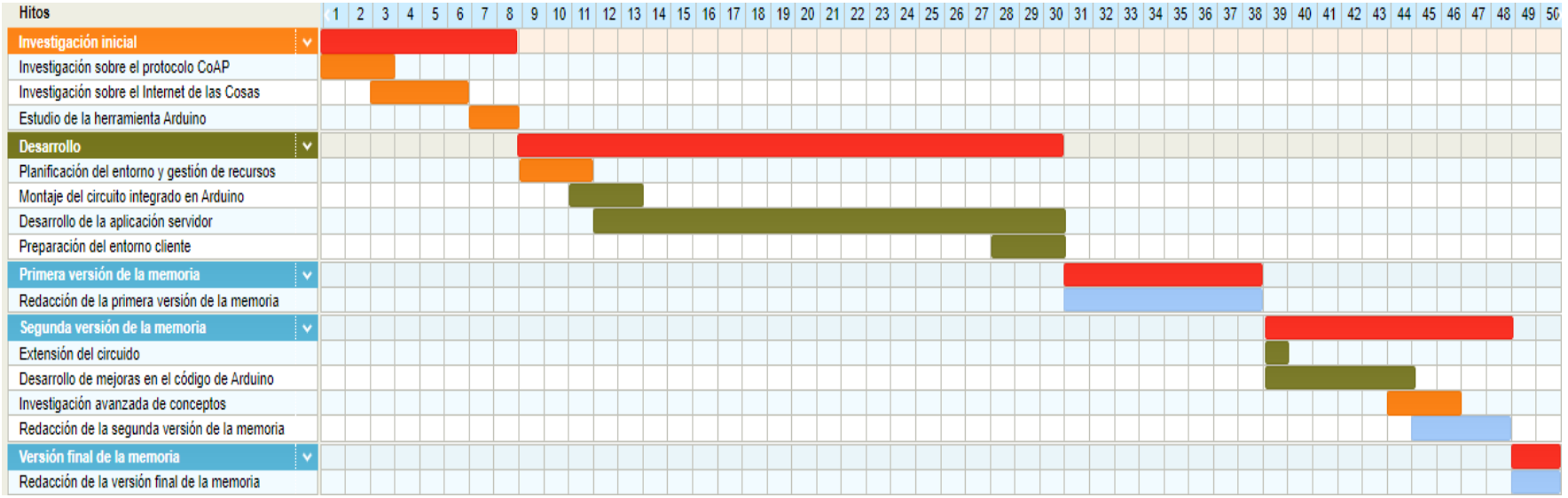

Figura 21: Diagrama de Gantt

(*) Los colores indican la tarea desempeñada:

- Rojo: Total del hito
- Naranja: Investigación
- Verde: Desarrollo
- Azul: Redacción de la memoria

Cada semana del diagrama implica un desempeño medio de 10 horas, teniendo en cuenta que se han desempeñado varias tareas en paralelo, dividiendo las horas semanales entre las tareas.

A partir de las tareas descritas en el diagrama de Gantt, se puede segregar el desarrollo del Trabajo de Fin de Grado en 5 fases o hitos:

- Una fase inicial correspondiente al estudio de las tecnologías del proyecto.

- La segunda fase comprende la ejecución de la parte práctica del proyecto, desde la planificación del entorno hasta la programación e integración de la aplicación.

- La tercera fase corresponde a la elaboración de la primera versión de la memoria.

- La cuarta fase hace referencia a la expansión de la parte práctica y la elaboración de la segunda versión de la memoria.

- La última fase corresponde a la ejecución de las correcciones y la elaboración de la versión final de la memoria.

Teniendo en cuenta el total de horas dedicadas al Trabajo de Fin de Grado, se obtiene un coste de 6000 €.

Por último, hay que contar también con las labores de dirección y supervisión de la tutora del proyecto, con un coste de 150 €

Teniendo en cuenta cada uno de los costes anteriores, se concluye que el presupuesto total del proyecto es de 7355.51 €.

6.2. Impacto socio-económico

El sector al que está enfocado el protocolo CoAP es el de los dispositivos IoT y M2M, los cuales en su mayoría no poseen la potencia necesaria para trabajar sobre HTTP. Como el propio nombre de CoAP indica, se trata de un protocolo creado especialmente para dispositivos con recursos limitados y ahí es donde puede ofrecer mejores prestaciones que HTTP, gracias a la escasa cantidad de recursos que necesita y la velocidad con la que realiza las tareas.

Si bien es verdad que se trata de un protocolo joven, el sector en el que se encuentra está en constante crecimiento, tanto que se han comenzado a proponer modificaciones sobre el protocolo, las cuales se han descrito en el apartado de estándares técnicos. Originalmente, el estándar de CoAP surgió en junio de 2014, mientras que las extensiones se han ido desarrollando a lo largo de los últimos años (RFC 7641 - septiembre 2015, RFC 7959 - agosto 2016, RFC 8323 - febrero 2018).

Estas modificaciones suponen que el protocolo CoAP ha conseguido un impacto social importante dentro del sector de las TIC.

En cuanto al impacto medioambiental, la implantación de este protocolo utilizando el sistema de sensores descrito en la parte práctica, podría permitir la automatización de procesos de supervisión que anteriormente requerían de la presencia de un técnico y que por su localización no fuese viable hacer un control de los mismos. La utilización de sensores para monitorizar las condiciones climáticas en estos lugares inaccesibles, o los sistemas de riego inteligentes, son ejemplos perfectos de cómo la tecnología puede ayudar al medio ambiente.

A pesar de la importancia de los dos factores anteriores, el que más mueve el mundo es sin duda el dinero. La prestigiosa revista Forbes lleva publicando artículos sobre el impacto económico del IoT a lo largo de los últimos años. Entre algunos de los casos [31], apuntó en el ahorro que supone el uso de esta tecnología para automatizar los procesos de fabricación en industrias o, el control y planificación de rutas en tiempo real para vehículos con su respectiva optimización del consumo de recursos.

La revista informa de que la gran mayoría de las empresas no son conscientes de las oportunidades de crecimiento que podrían obtener al utilizar este tipo de tecnologías. Esto indica que fuera del campo de las TIC, e incluso dentro de él, hay un desconocimiento elevado sobre estas nuevas ciencias pero que, poco a poco, se irá implementando en todos los sectores hasta convertirse en una tecnología de uso habitual. Una vez llegado a este punto es lógico pensar que se impondrán protocolos como CoAP para hacer uso de ellas.

En el caso concreto de este proyecto, la implantación del mismo supondría un ahorro en el consumo de energía y una respuesta más rápida de los dispositivos. El ahorro en el consumo de energía conllevaría a su vez a que los dispositivos a utilizar no tuviesen que tener unas características infladas como lo eran para HTTP, lo que se traduce en componentes que son más baratos.

7. CONCLUSIONES Y TRABAJOS FUTUROS

7.1. Objetivos cumplidos

En este apartado se pretende exponer las competencias adquiridas a raíz del desarrollo del presente Trabajo de Fin de Grado.

- Aunque se tuviese conocimientos previos de Arduino, estos no eran suficientes para la elaboración de la parte práctica y, al mismo tiempo que se ha ido implementando el código de las aplicaciones, se ha ido obteniendo mayores conocimientos respecto a esta herramienta, la manera en la que funciona, y el lenguaje de programación de la misma.

- En relación con el punto anterior, se ha podido enfrentar a la situación de tener que diseñar un escenario para un proyecto individual sobre el que trabajar desde cero. Con esta labor se ha logrado mejorar las dotes de iniciativa y resolución de problemas.

- El estudio del estado del arte ha permitido descubrir en profundidad la revolución que se aproxima en torno al Internet de las Cosas, la importancia del desarrollo de este sector y los distintos agentes que se encuentran en el mismo.

- Se ha obtenido un profundo conocimiento sobre el protocolo CoAP, su historia, su arquitectura e implementación práctica del mismo.

7.2. Conclusiones

En función de la investigación realizada y los resultados obtenidos en la evaluación práctica, se puede concluir lo siguiente:

- En términos de tiempo de respuesta del servidor, el protocolo CoAP es 10 veces más rápido que el protocolo HTTP. Teniendo en cuenta los datos obtenidos en el caso práctico, y basándose en los resultados del artículo de referencia, esta característica se mantiene independientemente de la carga bytes en el intercambio de mensajes.

- El número de bytes transmitidos durante el intercambio de los mensajes en el protocolo CoAP, si bien principalmente depende del desarrollo de la aplicación, se mantiene por debajo del de HTTP. Esto se debe al reducido tamaño de la cabecera de CoAP y al uso de UDP frente a TCP, el cual también tiene una cabecera más pequeña.

- El consumo de energía en cualquiera de las funciones de CoAP es menor al del protocolo HTTP. Al igual que en la cantidad de bytes, la fiabilidad que

aporta el uso de TCP también influye en el aumento del consumo de energía. Se confirma que el consumo está directamente relacionado con la cantidad de bytes transmitidos, a mayor tamaño, mayor es el consumo, y viceversa.

- Por su parte, las distintas variantes de uso de CoAP sacrifican el rendimiento en alguna característica a cambio de obtener una funcionalidad distinta al intercambio básico, pero siempre con mejores prestaciones globales que el protocolo HTTP.

7.3. Líneas futuras de trabajo

El propósito de esta sección es describir las futuras mejoras que se podrían realizar en el desarrollo de las aplicaciones si se dispusiese los recursos necesarios. Estos recursos hacen referencia principalmente al tiempo y al presupuesto.

En primer lugar, si bien es cierto que los dispositivos y componentes utilizados eran más que suficientes para realizar las pruebas pertinentes, se habría preferido tener un mayor presupuesto para comprar elementos más precisos y con mayores prestaciones. Un caso concreto es el del módulo WiFi, el cual podría haber sido sustituido por un componente que proporcione acceso a internet pero que además tuviese la posibilidad de implementar el protocolo IPv6. Otra variante del circuito habría sido utilizar un osciloscopio en lugar del sensor de intensidad para poder obtener unos datos exactos del consumo de energía en función de la labor desempeñada por el servidor.

La limitación que ha surgido por parte de la librería libcoap es que la misma no permitía el uso de envíos seguros sobre CoAP, es decir, del protocolo DTLS por medio de la query de “coaps”. Una mejora sería implementar los casos prácticos con este tipo de intercambio para poder comparar las prestaciones del protocolo cuando se transmite de forma segura y cuando no.

Otra posibilidad sería realizar las modificaciones pertinentes en la aplicación para evaluar los cambios propuestos en el RFC 8323 correspondiente a la transmisión de CoAP por medio de TCP, TLS y WebSockets.

Por último, y quizás la mejora más interesante, sería poder implementar otros protocolos de IoT, como MQTT o XMPP, sobre el escenario empleado en este proyecto y comparar los resultados prácticos con los obtenidos para el protocolo CoAP.

7.4. Reflexiones personales

Durante el desarrollo de este Trabajo de Fin de Grado, se ha podido comprobar la magnitud de la expansión de internet y lo que ello conlleva. El Internet de las Cosas es igual a futuro, y como todo futuro, esto significa progresar. Al igual que IP se ha

adaptado a las nuevas condiciones del mercado, los protocolos de nivel de aplicación deben evolucionar y prepararse para lo que está por venir.

En el caso concreto de las redes de sensores o WSN, se ha comprobado por medio de los resultados obtenidos en los casos prácticos, que el protocolo CoAP supera en todos los sentidos las prestaciones que ofrece el protocolo HTTP. Ya no sólo utilizando los mismos métodos que posee HTTP, sino que además añade nuevas formas de trabajar sobre los recursos por medio de funcionalidades como la observación o el descubrimiento de recursos.

Se trata de un protocolo creado especialmente para nodos y redes de recursos restringidos, es lógico por tanto, que el rendimiento del mismo sea mejor en este campo. La ventaja de CoAP es que se trata de un protocolo no invasivo, es decir, que se puede implementar sobre la arquitectura existente en internet sin necesidad de modificarla, al igual que tiene la posibilidad de interactuar con el protocolo HTTP.

El crecimiento de CoAP ha sido constante durante los últimos años, ofreciendo cada vez una mayor variedad de implementaciones y servicios que se van adaptando a los requisitos de las nuevas tecnologías. CoAP posee las herramientas necesarias para lograr automatizar y autonomizar las redes de sensores, llegando un día a prescindir de la intervención humana en el proceso.

8. BIBLIOGRAFÍA

[1] J. M. Batalla, et al. *Beyond the Internet of Things Everything Interconnected*. Cham: Springer International Publishing, 2017.

[2] W. Stallings, *Foundations of Modern Networking: SDN, NFV, QoE, IoT, and Cloud*, 1st ed. Addison-Wesley Professional, 2015.

[3] S. Hagen, *IPv6 Essentials*, 2nd ed. Sebastopol: O'Reilly Media, 2009.

[4] Z. Shelby y C. Bormann, *6LoWPAN: The Wireless Embedded Internet*. Chichester: J. Wiley, 2009.

[5] G. C. Hillar, *MQTT essentials - a lightweight IoT protocol: The preferred IoT publish-subscribe lightweight messaging protocol*. Birmingham; Mumbai: Packt Publishing, 2017.

[6] P. Saint-André, K. Smith y R. Troncon, *XMPP*. O'Reilly Media, Inc, 2009.

[7] *The Constrained Application Protocol (CoAP)*, Internet Engineering Task Force RFC 7252-2014.

[8] *Constrained RESTful Environments (CoRE) Link Format*, Internet Engineering Task Force RFC 6690-2012.

[9] *Constrained RESTful Environments (CoRE) Parameters*, Internet Assigned Numbers Authority.

[10] W. Colitti, K. Steenhaut, N. De Caro, B. Buta, y V. Dobrota, "Evaluation of constrained application protocol for wireless sensor networks," presentada en 18th IEEE Workshop on Local & Metropolitan Area Networks (LANMAN), Chapel Hill, 13-14 oct., 2011. [En línea]. Disponible en: https://ieeexplore.ieee.org/document/6076934/

[11] *Observing Resources in the Constrained Application Protocol (CoAP)*, Internet Engineering Task Force RFC 7641-2015.

[12] D. Wheat, *Arduino Internals*. Berkeley: Apress, 2011.

[13] B. Earl, “You know you have a memory problem when...”, *adafruit*. [En línea]. Disponible en: https://learn.adafruit.com/memories-of-an-arduino/

[14] Foro de Arduino. [En línea]. Disponible en: https://forum.arduino.cc/index.php?topic=496854.0

[15] Mouser Electronics, “DHT 11 Humidity & Temperature Sensor”. [En línea]. Disponible en:

https://www.mouser.com/ds/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf

[16] Espressif Systems, “ESP8266EX Datasheet”. [En línea]. Disponible en: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf

[17] Allegro MicroSystems, “ACS712-Datasheet”. [EN línea]. Disponible en: https://www.allegromicro.com/~/media/files/datasheets/acs712-datasheet.ashx

[18] P. Stoffregen, “TimerThree”, *GitHub*. [En línea]. Disponible en: https://github.com/PaulStoffregen/TimerThree

[19] M. Ruys, “arduino-DHT”, *GitHub*. [En línea]. Disponible en: https://github.com/markruys/arduino-DHT

[20] ITEAD Studio, “ITEADLIB_Arduino_WeeESP8266”, *GitHub*. [En línea]. Disponible en: https://github.com/itead/ITEADLIB_Arduino_WeeESP8266

[21] DevicePilot, “microcoap”, *GitHub*. [En línea]. Disponible en: https://github.com/1248/microcoap

[22] *i2010: la sociedad de la información y los medios de comunicación al servicio del crecimiento y el empleo*, Comisión Europea, 2005.

[23] Directiva 2014/53/UE del Parlamento Europeo y del Consejo, de 16 de abril de 2014, relativa a la armonización de las legislaciones de los Estados miembros sobre la comercialización de equipos radioeléctricos, y por la que se deroga la Directiva 1999/5/CE.

[24] Equipo de Internet de las Cosas, “The Alliance for Internet of Things Innovation (AIOTI)”, Comisión Europea, 13-05-2015. [En línea]. Disponible en: https://ec.europa.eu/digital-single-market/en/alliance-internet-things-innovation-aioti

[25] Documento de trabajo del personal de la Comisión Europea, de 19 de abril de 2016, relativo al avance del Internet de las Cosas en Europa.

[26] “Connected Communities Initiative”, Comisión Europea, 18-05-2015. [En línea]. Disponible en: https://ec.europa.eu/digital-single-market/en/news/connected-communities-initiative

[27] *Terminology for Constrained-Node Networks*, Internet Engineering Task Force RFC 7228-2014.

[28] *Group Communication for the Constrained Application Protocol (CoAP)*, Internet Engineering Task Force RFC 7390-2014.

[29] *Block-Wise Transfers in the Constrained Application Protocol (CoAP)*, Internet Engineering Task Force RFC 7959-2016.

[30] *CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets*, Internet Engineering Task Force RFC 8323-2018.

[31] L. Columbus, “Where IoT Can Deliver The Most Value In 2018”, *Forbes*, marzo 2018. [En línea]. Disponible en: https://www.forbes.com/sites/louiscolumbus/2018/03/18/where-iot-can-deliver-the-most-value-in-2018/

ANEXOS

Anexo A: Componentes utilizados en los casos prácticos

- Arduino UNO rev3:

Figura A.1: Arduino UNO rev3

- Arduino MEGA 2560 rev3:

Figura A.2: Arduino MEGA 2560 rev3

- Módulo WiFi ESP8266-01:

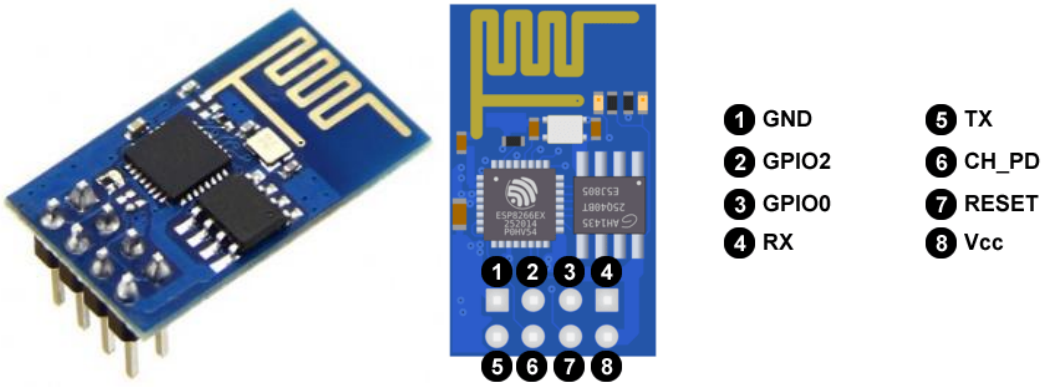

Figura A.3: Módulo WiFi ESP8266-01

- Sensor de temperatura y humedad DHT11:

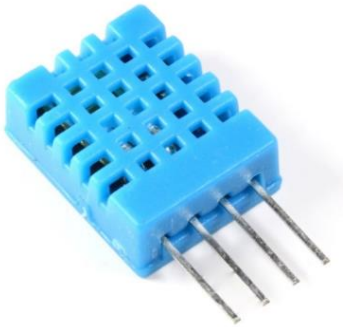

Figura A.4: Sensor de temperatura y humedad DHT11

- Sensor de corriente ACS712-30A:

Figura A.5: Sensor de corriente ACS712-30A

- Placa protoboard:

Figura A.6: Placa protoboard

- Cables DuPont:

Figura A.7: Cables DuPont hembra-hembra, macho-hembra y macho-macho

- Resistencias:

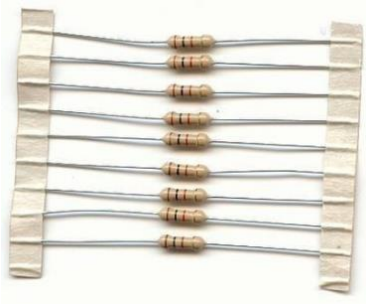

Figura A.8: Resistencias

- Multímetro PS-TECH PS-MB01

Figura A.9: Multímetro PS-TECH PS-MB01

Anexo B: Glosario

6LoWPAN IPv6 over Low-Power Wireless Personal Area Networks

AIOTI The Alliance for Internet of Things Innovation

CoAP Constrained Application Protocol

CoRE Constrained RESTful Environments

EEPROM Electrically Erasable Programmable Read-Only Memory

HTTP Hypertext Transfer Protocol

IETF Internet Engineering Task Force

IoT Internet of Things

IP Internet Protocol

IPv4 Internet Protocol version 4

IPv6 Internet Protocol version 6

ITU International Telecommunication Union

M2M Machine-to-Machine

MQTT Message Queuing Telemetry Transport

OASIS Organization for the Advancement of Structured Information Standards

REST Representational State Transfer

RFC Request For Comments

SRAM Static Random Access Memory

TCP Transmission Control Protocol

TIC Tecnologías de Información y Comunicación

UDP User Datagram Protocol

WSN Wireless Sensor Network

XML Extensible Markup Language

XMPP Extensible Messaging and Presence Protocol

Anexo C: Extended Abstract

Introduction

We live in a world in which tendency is to transform objects to work autonomously, without human intervention, becoming devices that can communicate with each other through the Internet. The Internet is made up of a set of networks that allows the decentralized interconnection of devices through a group of protocols. There are different models on the internet that follow a layer-based architecture, and each of them describes how the process of sending data through a network is.

The Internet of Things is the present and the future of the coming years in the ICT (Information and communications technology) sector and the need arises to study more suitable ways of communication between these devices. Although it is true that to adapt the communication to the IoT environment it would be necessary to optimize the internet at all levels, it is not so simple to make a change of the entire architecture. The focus is on application-level protocols since the main differences are manifested at this level, for example when processing the data and generating the packages, considering that the devices have limited resources in IoT.

This Final Degree Project focuses on the evaluation of the application-level protocol CoAP. CoAP is designed to be used in devices with limited resources such as wireless sensor networks (WSN). It aims to replicate the way in which HTTP works but designed to work over M2M and IoT applications. To evaluate the CoAP protocol, it will be compared with HTTP on the same Internet of Things scenario.

It is common that new protocols that aim to improve benefits during the exchange of messages appear. For this reason, a compilation of the current situation of the Internet of Things is also made, considering the different protocols that have rebounded more in recent decades, such as MQTT or XMPP.

State of the art

The Internet of Things (IoT) is a concept that refers to the interconnection of day-to-day objects with the internet. Even if the existence of autonomous interconnected devices is quite old (for example, ATMs appeared in 1967), the term Internet of Things didn’t appear until 1999 coined by Kevin Ashton. It is six years later, in 2005, when the ITU showed interest in this concept by making an exhaustive report about it.

The IoT can be found in many sectors, such as home automation; fast food chains, where devices notify when the food is ready; in logistics, tracking the place where an order is located at each moment; or in agriculture, with sensors of temperature, humidity, and other climatic factors. This last service is the one intended to be reproduced in the practical cases of this project.

The changes that have taken place on the Internet in the last decades led to a change in the structure of the Internet. Considering that the devices that work on the Internet of Things have limited characteristics in terms of power or bandwidth, the architecture must implement protocols that adapt to these restricted scenarios. Two levels have been affected: the network level and the application level.

The IPv6 network-layer protocol aims to replace its predecessor IPv4.The importance of IPv6 for the Internet of Things lies in 2 essential characteristics. First, it can identify up to 340 sextillions addresses since a great amount is needed for autonomous devices and it is impossible to visualize a future for IoT in a limited scenario such as IPv4. Secondly, IPv6 has greater security, incorporating encryption and authenticity verification of the origin of messages.

Applied to the Internet of Things, the standard 6LoWPAN (IPv6 over Low-Power Wireless Personal Area Network) emerges as an adaptation-layer that allows networks based on the IEEE 802.15.4 physical-layer standard to use the IPv6 protocol. The goal of this standard is to ease the interconnection of small devices with low power and limited resources.

The main interest of application-layer protocols is that there is where the encapsulation of the data takes place, and as its name indicates, they are generated in different ways depending on the use given to that data. The MQTT, XMPP, and CoAP protocols have been chosen due to their impact:

- MQTT: It is a protocol that works over TCP / IP and allows the encryption of confidential messages using a connection with TLS. It is designed to minimize the bandwidth and the use of resources thanks to extremely light message exchanges. The protocol was invented by Andy Stanford-Clark of IBM and Arlen Nipper of Arcom in 1999, becoming an OASIS standard in 2014.

  This protocol is based on a publish-subscribe system. In that exchange, the publishers, which means the senders of the messages, don't program the message to be sent to the receiver, but instead, they categorize the messages in different topics without knowing the subscribers to whom they will be sent. In the same way, the subscribers show interest in different topics without knowing the origin of them.

  This protocol requires a *broker*, which is a central node that performs server tasks to manage the exchange of the topics of interest from the publishers to the subscribers.

- XMPP: This protocol originally emerged as a means for the exchange of instant messages like chat services. This protocol was created by Jeremie Miller in the year 1999 with the name of "Jabber", although currently, this name is in disuse. It is an XML-based protocol and, like HTTP or MQTT, works over TCP / IP.

  In terms of architecture, the protocol has 3 elements: XMPP clients, XMPP servers and gateways. The XMPP clients can’t contact each other by

themselves and need the XMPP servers to make the exchange of messages. If an XMPP client wants to communicate with a client that doesn't use the XMPP protocol, the XMPP servers route the messages to a gateway that will translate the message to a different protocol supported by the target client.

The operation of XMPP is based on two fundamental concepts: XML Streams and XML Stanzas. First, the XML Streams must establish the connection between two entities and, once established, the XML Stanzas are sent. Depending on the mode in which the connection is configured, a type of Stanza XML will be sent: request/response (iq), publication/subscription (presence) and asynchronous message (message).

- CoAP: CoAP works on limited environments following the CoRE (Constrained RESTful Environments) standard of the IETF, which provides a framework for resource-oriented applications and aims to implement the REST architecture on restricted nodes (for example, 8-bit microcontrollers with limited RAM and ROM) and networks (for example, 6LoWPAN). The CoRE architecture consists of nodes that are responsible for one or more resources, such as sensors or state controllers, and are responsible for the exchange of messages.

  In addition, CoRE allows discovering the resources managed by a device. To achieve this, the client makes a request including the text "/.well-known/core" to the destination server, which will respond with a summary of the resources it manages, but not the values of those resources. Thanks to this message, the client can generate the URI to make the request for a specific resource from the server.

  The purpose of the CoAP design is to keep the message length small, avoiding the need of fragmentation. This doesn't mean to blindly compress HTTP, but rather to make a subset of REST (POST, GET, PUT and DELETE) common to HTTP optimized for machine-to-machine applications. Therefore, one of the main goals of CoAP is to obtain a generic web protocol that meets the requirements of restricted environments.

Although MQTT with the publish/subscribe model and the different Stanzas offered by XMPP are two interesting variants, the CoAP protocol has been chosen as the central node of this Final Degree Project. This choice is due to the variety of functionalities offered by the CoAP protocol, maintaining a scheme known as it is the REST architecture.

CoAP protocol

CoAP defines four types of messages: Confirmable (CON), Non-Confirmable (NON), Acknowledgement (ACK) and Restart (RST). The requests can be sent via Confirmable or Non-Confirmable messages while the response can be sent in Non-Confirmable, Acknowledgment or Restart messages.

The reliable transmission option is achieved by choosing the message as Confirmable. A Confirmable message is transmitted using a default timeout and an exponential forward time between successive retransmissions until the receiver sends an Acknowledgement message with the same identifier and it reaches the sender. Any message that does not require a reliable transmission can be sent as a Non-Confirmable. These messages are answered with another Non-Confirmable message. When the receiver is not able to fully process the request message, it responds with a Restart message instead of an Acknowledgement or Non-Confirmable message.

The Method Codes and Response Codes in some of these messages are responsible for making them take requests or answers. These Codes have different values including the REST architecture.

The message format starts with a 4-byte header formed by the version, the type, the Token Length, the Code and the Message ID. The Message ID is used to detect the duplicated messages and to match the response to its request. After the header can go the optional fields: the Token (correlate the requests and responses), the Options and/or the Payload.

Unlike HTTP or MQTT, CoAP uses the UDP protocol in the transport layer, which means asynchronous interchange of messages and allows multicast. While HTTP is secured using Transport Layer Security (TLS) over TCP, CoAP is secured using Datagram TLS (DTLS) over UDP. This feature is optional in CoAP and can be selected by using the “coaps” Uri-scheme instead of the no-sec “coap”. Also, the multicast exchange doesn’t enable the secure mode in the messages.

Practical evaluation

The starting point of the Final Degree Project was the article "Evaluation of Constrained Application Protocol for Wireless Sensor Networks", which compares between the protocol CoAP and HTTP.

The main goal of this Final Degree Project was to reproduce the comparison between protocols made in the article but using a simple physical circuit in the environment of the Internet of Things, avoiding the simulation of any of the characteristics and basing all the results on real data. In addition to the main objective, some features of the CoAP protocol has been studied, including the reliability, discover and observe functions.

To carry out the tests, it has been decided to use a tool with many possibilities and a projection in the world of IoT. This tool is Arduino. Arduino is an open source free hardware platform based on a board with a microcontroller and a development environment, designed to facilitate the use of electronics in multidisciplinary projects.

The software part of Arduino consists of an integrated development environment (IDE). It is a computer program consisting of a code editor, a compiler, a debugger, a graphical user interface (GUI) and the tools necessary to load the compiled code into the hardware's flash memory.

The sketches refer to the program itself, the place where the code of the application is edited and subsequently compiled and executed on the Arduino board. The language used to program is an own language that is based on C ++ but applied to microcontrollers. The sketch has two basic elements that are mandatory in all programs: setup and loop.

```
void setup() {
    initializeSerial(); // initialize Serial and Serial3 at 115200
    Baud rate
    setupESP8266(); // connect and configure the ESP-01
    dhtSensor.setup(DHT_DATA_PIN); // set the input pin
    unsigned long period = dhtSensor.getMinimumSamplingPeriod()*1000;
    Timer3.initialize(period); // initialize timer3, and set the
    minimum period
    Timer3.attachInterrupt(updateTime); // attaches function as a
    timer overflow interrupt
    Serial.println("ready");
}

void loop() {
    refreshDHTSensor(); // refresh every 'period' time
    listenRequest();
}
```

Figure D.1: Setup and loop methods in Arduino's code

In the scenario of the project, within the setup takes place the configuration of the Serial, the connection to the Internet by means of the ESP8266 WiFi module and its operation mode, the pin that reads the input of the DHT and the timer frequency. The loop is responsible for updating the values of the sensor and listening if a request has been received on any resource, in which case a response is sent.

The hardware part is based on a microcontroller, which means a programmable integrated circuit with the ability to execute the instructions recorded in its memory. The microcontroller is composed of three functional units that are: a central processing unit, better known as CPU, memory, and input and/or output peripherals.

Arduino has a wide variety of versions of its own hardware adapted for the different types of requirements that the user needs. For this project, the Arduino Mega board has been chosen. Additionally, there are three components that have been used:

- DHT11: temperature (from 0ºC to 60ºC with accuracy of 1ºC) and humidity (from 0% to 60% with an accuracy of 5%) sensor.

- ESP8266-01: integrated microchip with WiFi connection and compatible with the different protocols of the TCP/IP model.

- ACS712-30A: current sensor with a range from -30 A to 30 A.

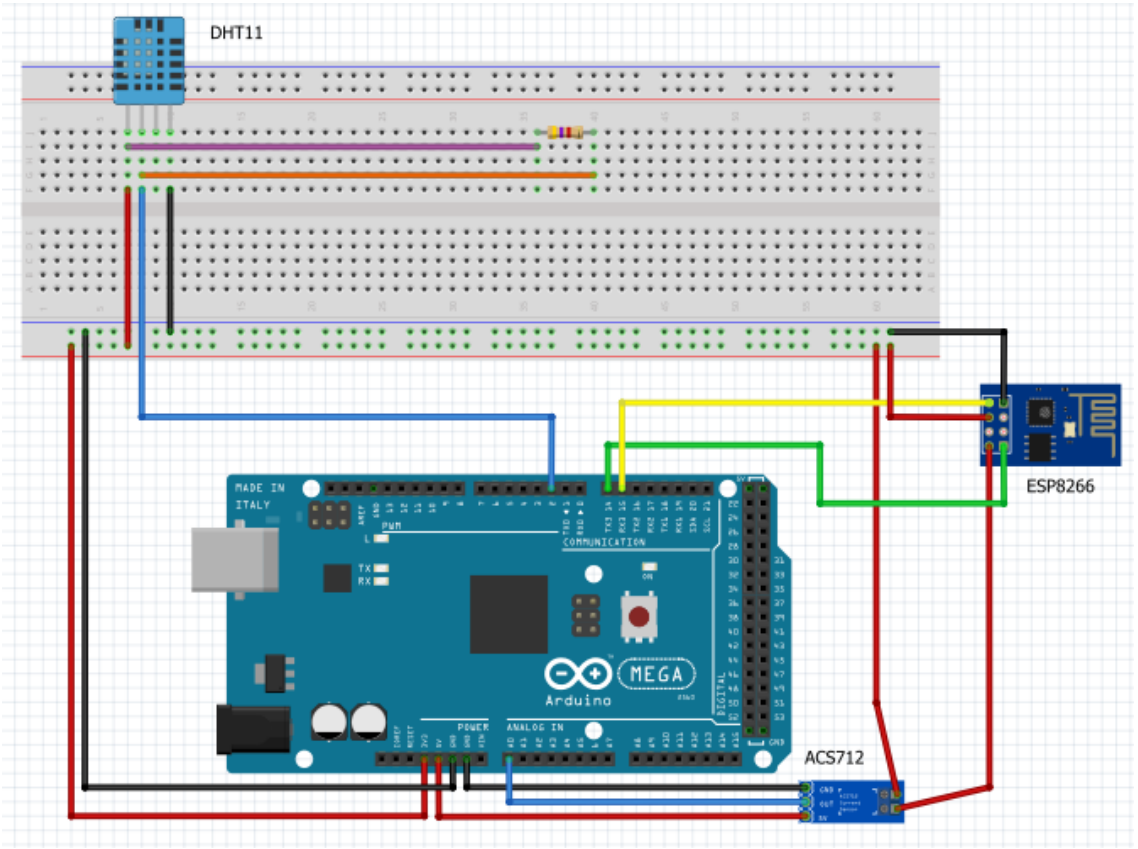

Figure D.2: Physical circuit of the server

Starting from the client side, and to simulate the requests, libcoap libraries were used for the CoAP protocol and libcurl for the HTTP protocol.

As it was commented previously, the objective was to compare a simple exchange of request/response messages in an IoT environment. The main goal was to study the behavior of CoAP and HTTP in the same situation and additionally the next functionalities of CoAP were also added:

- The difference in the exchange when it is carried out in a reliable and unreliable message, that is, when a Confirmable message is used in the request and when a Non-Confirmable message is used. In this way, it is possible to check if the same protocol offers different services during the delivery depending on the reliability of the exchange.

- The feature of discovering the resources managed by a server using the CoRE architecture that implements the CoAP protocol. It is a useful mechanism in the Internet of Things sector for its ability to autotomize devices and allow an M2M communication without a human intervention.

- The option "*observe*" that allows observing a specific resource managed by a server. This option did not exist in the original approach of the protocol and it is an extension of it. The message exchange system that uses this option is like the publish/subscribe method described in the MQTT part, but without using a *broker* for it.

Conclusions

- In terms of server response time, the CoAP protocol is 10 times faster than the HTTP protocol. Considering the data obtained in the practical evaluation, and based on the results of the reference article, this characteristic is maintained independently of the length of the messages in the exchange.

- The number of bytes transmitted during the exchange of messages in the CoAP protocol, although mainly depend on the development of the application, remains below the amount in HTTP. This is due to the small size of the CoAP header and the use of UDP over TCP, which also has a smaller header.

- The power consumption in any of the CoAP functions is lower than in the HTTP protocol. As in the number of bytes, the reliability provided using TCP also influences the increase in energy consumption. It is confirmed that the consumption is directly related to the number of bytes transmitted, the larger the size, the greater the consumption, and vice versa.

- On the other hand, the different variants of CoAP sacrifice the efficiency in some feature in the exchange for obtaining a functionality different from the basic exchange, but always with better global benefits than the HTTP protocol.